Exploración visual basada en similitud de grandes conjuntos de datos multidimensionales georreferenciados

Eddie Rogger Peralta Aranibar

Orientador: Dr. Erick Gomez Nieto

Jurado:
Dr. Luis Gustavo Nonato (Universidade de São Paulo - Brasil)
Dr. Hélio Pedrini (Universidade Estadual de Campinas - Brasil)
Dr. Alex Cuadros Vargas (Universidad Católica San Pablo - Perú)
Dr. José Eduardo Ochoa Luna (Universidad Católica San Pablo - Perú)

Tesis presentada al
Departamento de Ciencia de la Computación
como parte de los requisitos para obtener el grado de
Maestro en Ciencia de la Computación.

Universidad Católica San Pablo – UCSP
Abril de 2019 – Arequipa – Perú

A toda mi familia por haberme brindado su apoyo incondicional, en especial a ti madre, lo logramos. A mis abuelos José y Emma que desde el cielo me deben estar viendo.

Abreviaturas

AKNN *Approximate kernel K-Nearest Neighbors*

CPU *Central Processing Unit*

CSV *comma-separated values*

CUDA *Compute Unified Device Architecture*

DBMS *Database Management Systems*

GPU *Graphics Processor Unit*

KNN *k-nearest neighbors*

NYC *New-York City*

OLAP *On-Line Analytical Processing*

PCA *Principal Component Analysis*

RAM *Random-access memory*

STIG *Spatio-Temporal Indexing using GPUs*

TLC *New York City Taxi and Limousine Commission*

t-SNE *t-Distributed Stochastic Neighbor Embeddin*

Agradecimientos

Primero que todo me gustaría agradecer a Dios por guiarme, iluminarme y darme tranquilidad para seguir luchando por mis objetivos y no desanimarme con las dificultades que fueron surgiendo a lo largo de mi recorrido.

Agradezco también a mis padres, Leonor Aranibar Motta y Eddie Peralta Ramiréz, que siempre me motivaron, quieren y querrán lo mejor para mí, que con su esfuerzo y dedicación me ayudaron a alcanzar mis metas y me dieron el apoyo suficiente para no decaer cuando todo parecía complicado, gracias a ella he llegado a culminar un peldaño más de mi vida.

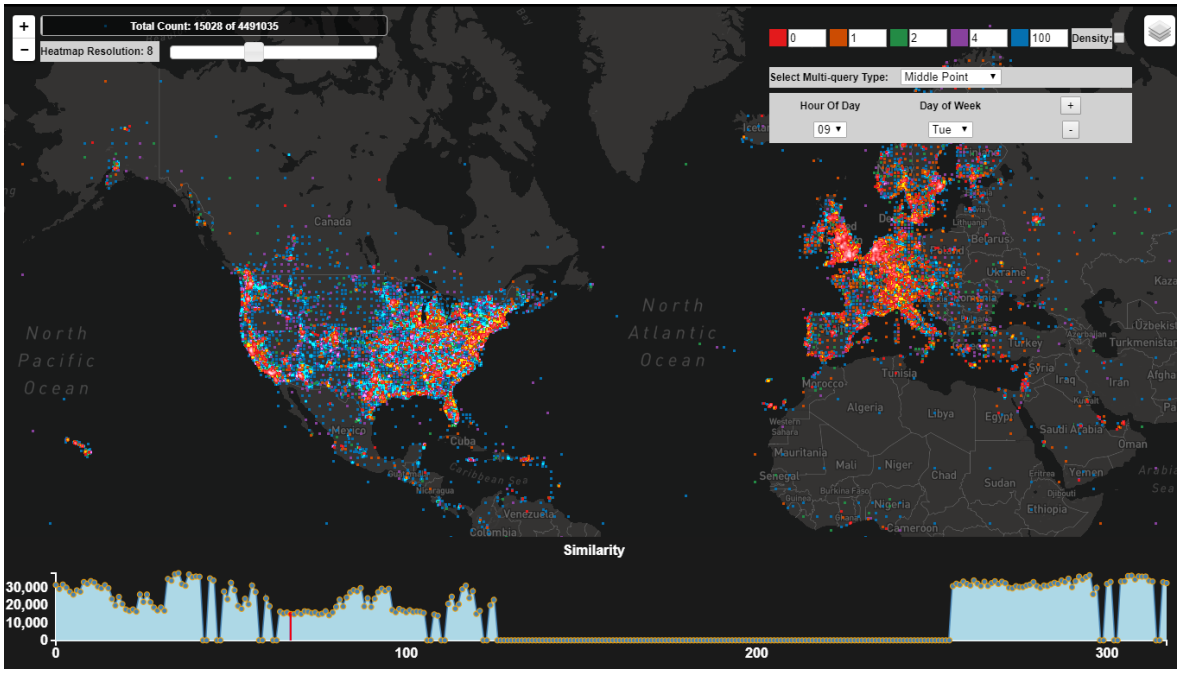
A todos mis tíos, Rosemary por darme su amor como un hijo más, Huber por su gran ejemplo moral, Ronald, Arturo, Juan y a mis padrinos Cesar Pacheco y Nalddi Aranibar, porque con sus oraciones, consejos y palabras de aliento hicieron de mí una mejor persona y de una u otra forma me acompañan en todos mis sueños y metas.

A todos los amigos que de una forma directa e indirectamente, contribuyeron o me ayudaron en la elaboración del presente estudio, en especial a Gina Lucia Muñoz Salas, por la atención, paciencia y empeño que me ofreció en cada momento que lo necesité.

A el Profesor Doctor Erick Gómez Nieto, por su orientación, total apoyo, disponibilidad, por sus opiniones, criticas y total colaboración en solucionar dudas y problemas que han ido surgiendo a lo largo de la realización de este trabajo. Y sobre todo por sus palabras de incentivo y empuje.

Deseo agradecer de manera especial al Consejo Nacional de Ciencia, Tecnología e Innovación Tecnológica (CONCYTEC) y al Fondo Nacional de Desarrollo Científico, Tecnológico e Innovación Tecnológica (FONDECYT-CIENCIACTIVA), que mediante Convenio de Gestión 234-2015-FONDECYT, han permitido la subvención y financiamiento de mis estudios de Maestría en Ciencia de la Computación en la Universidad Católica San Pablo (UCSP).

Abstract



Big data visualization is a main task for data analysis. Due to its complexity in terms of volume and variety, very large datasets are unable to be queried for similarities among entries in traditional Database Management Systems. In this thesis, we propose an effective approach for indexing millions of elements with the purpose of performing single and multiple visual similarity queries on multidimensional data associated with geographical locations. Our approach makes use of the method Z-Curve method to map into 1D space considering similarities between data. We support our proposal by comparisons with state-of-the-art methods in the literature, using neighborhood preservation methods and analyzing the advantages and disadvantages of these methods. Additionally, we present a set of results using real data of different sources and we analyze the insights obtained from the interactive exploration.

Keywords: Visualization, Similarity, Interactive Visualization, Data cubes.

Resumen

La visualización de grandes cantidades de datos es una de las principales tareas que realiza un analista de datos. En sistemas tradicionales de manejo de datos, registros de enormes conjuntos de datos no pueden ser consultados por su similitud debido a su complejidad, en términos de volumen y multiplicidad. En esta tesis, proponemos un enfoque efectivo para la indexación de millones de elementos, con el propósito de ejecutar simples y múltiples consultas visuales de similitud sobre datos multidimensionales asociadas a una ubicación geográfica. Nuestro enfoque hace uso del método *Z-order curve* para mapear nuestro conjunto de datos en una alta dimensionalidad a un espacio de una dimensión considerando la similitud entre los datos. Respaldamos nuestra propuesta mediante la comparación con otros métodos del estado del arte en la literatura, utilizando métricas de preservación de vecindad y analizando las ventajas y desventajas entre estos métodos. Adicionalmente, presentamos un conjunto de resultados usando datos reales de diversas fuentes y analizamos los conocimientos obtenidos a partir de su exploración interactiva.

Keywords: Visualización, Similitud, Visualizació interactiva, Data cubes.

Índice general

Índice de cuadros

Índice de figuras

Capítulo 1

Introducción

La exploración y visualización de datos se ha convertido en una tarea cada vez más compleja de lograr con el paso del tiempo. Esto es debido a su rápido crecimiento; por ejemplo, si se desea recuperar registros arbitrarios de una base de datos geolocalizada de gran tamaño, necesitaremos algoritmos no-triviales paralelos, o por lo contrario bastante tiempo para producir un gráfico. Además de ser lo suficientemente grande para caber en la memoria, requiriendo ser distribuida sobre varios *clusters*.

Gracias a la simplicidad en el uso de varias aplicaciones masivas de datos (redes sociales, mercado electrónico o sistemas de información greográficos), el tratamiento de grandes volúmenes de datos ha adquirido gran popularidad. Por lo que, el manejo adecuado de su tiempo de procesamiento y requerimiento de memoria han sido una de las mayores preocupaciones para los *Database Management Systems* (DBMS).

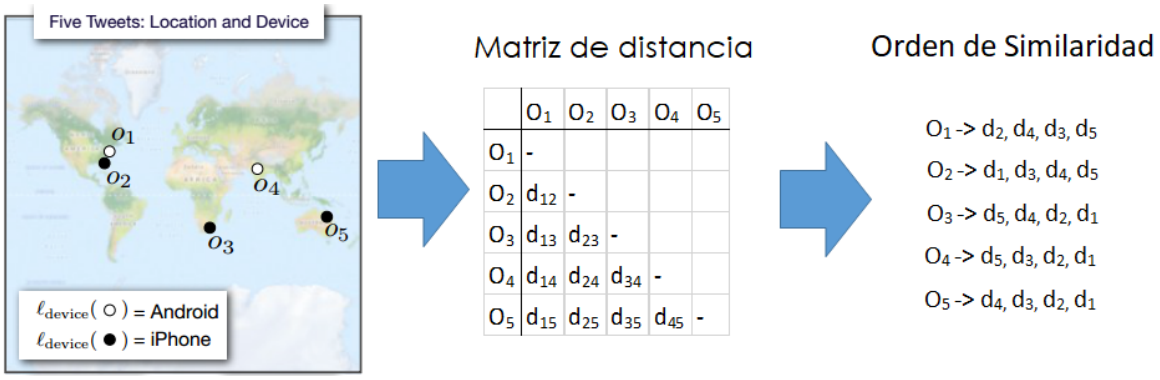
Sin embargo, los operadores para la extracción de información, como consultas de similitud, no pueden ser realizadas en sistemas relacionales (PostgreSQL, MySQL), NoSQL (MongoDB, Cassandra) o basados en grafos (Neo4j, OrienDB) forzando al analista de emplear herramientas adicionales o implementar sus propios operadores para obtener resultados similares.

1.1. Motivación y contexto

Encontrar las relaciones de similitud que presentan grandes volúmenes de datos es una tarea difícil para el análisis de datos, en general, debido al cálculo costoso de las similitudes entre un alto número de entradas y es un requerimiento en muchas aplicaciones como las siguientes: encontrar páginas con significado similar para su clasificación o para evitar páginas web duplicadas, encontrar usuarios con gustos parecidos con el fin de desarrollar un sistema de sugerencias, etc. En particular, esta búsqueda de vecinos cercanos debido a la maldición de la dimensionalidad es un proceso costoso de llevar a cabo (Beyer et al., 1999) (Böhm et al., 2001). En este contexto, la complejidad

del cálculo es totalmente dependiente de dos características cruciales: la medida de la distancia y la dimensionalidad de los datos. Es esencial considerar la medida de la distancia, ya que dependiendo del tipo de datos, se elegirá una medida de distancia adecuada, por ejemplo, distancias Euclidean y Manhattan para datos numéricos, distancia Jaccard para datos categóricos y distancia Gower para datos mixtos. Lógicamente, todas las medidas mencionadas anteriormente difieren en su formulación, mientras que la complejidad de los tipos de datos aumenta. En segundo lugar, las aplicaciones modernas se utilizan con una gran cantidad de atributos para describir con precisión los objetos, lo que tiene un impacto dramático en el cálculo del costo de similitud. Estos inconvenientes importantes impiden la adición de consultas de similitud en el contexto de los DBMS.

A pesar de la existencia de diferentes métodos de indexación propuestos para la reducción de la latencia, tales como la estructura de datos KD-tree, dichos enfoques no son más eficientes que la utilización exhaustiva de fuerza bruta, cuya complejidad es $O(nD)$ de un conjunto finito de n vectores $x \in \mathbb{R}^D$. La figura 1.1 muestra el resultado de los pasos seguidos para obtener un orden por similitud, luego de obtener una matriz de distancia entre todos los datos.



	O_1	O_2	O_3	O_4	O_5
O_1	-				
O_2	d_{12}	-			
O_3	d_{13}	d_{23}	-		
O_4	d_{14}	d_{24}	d_{34}	-	
O_5	d_{15}	d_{25}	d_{35}	d_{45}	-

Figura 1.1: Ilustración de los pasos realizados para obtener un orden por similitud entre datos geo-referencidos.

Como mencionamos anteriormente, encontrar una determinada cantidad de objetos similares a otro dada consulta dada *k-nearest neighbors* (KNN), en una colección de datos de alta dimensionalidad es un problema difícil de resolver y proporcional al número de dimensiones.

La solución más simple, es una búsqueda lineal, que calcule la distancia de la consulta a cada objeto en el conjunto de datos y retorne los k más cercanos tiene una complejidad de $O(nd)$, donde n es el tamaño del conjunto de datos y d la dimensionalidad por lo que para grandes conjuntos de datos no sería viable.

Esta dificultad de encontrar los vecinos cercanos sobre una gran cantidad de datos de alta dimensionalidad ha dado desarrollo a algoritmos aproximados, conocido *Approximate kernel K-Nearest Neighbors* (AKNN).

A continuacón, evidenciamos las dificultades anteriormente mencionadas usando

Taxis NYC			
Entrada de datos	**CPU Secuencial (ms)**	**CPU Threads (ms)**	**GPU (ms)**
10000	3.152	0.631	0
11000	3.732	0.879	0
12000	4.418	0.924	0
13000	5.256	1.029	0
14000	6.036	1.213	0
15000	7.273	1.464	0
16000	7.904	1.56	0
17000	8.902	1.735	0
18000	9.986	1.926	0
19000	11.133	2.224	0
20000	12.316	2.374	0
30000	27.29	5.376	0
35000	37.1	7.167	0
40000	51.189	9.339	0
45000	64.469	11.865	0
50000	82.207	14.839	0
55000	96.266	17.766	0
60000	116.148	21.913	0

Cuadro 1.1: Comparación entre CPU secuencial, CPU paralelizado y procesamiento masivo en GPU. Para el conjunto de datos de taxis de NYC.

dos experimentos realizados para medir la complejidad en realizar los pasos en obtener una relación de similitud entre los datos (Figura 1.1). Además, describiremos en detalle los conjuntos de datos utilizados junto con los resultados obtenidos, La cantidad de objetos de entrada considerados pueden ser almacenados en una memoria *Random-access memory* (RAM) de 16 GB.

En el cuadro 1.1 se muestra una evaluación experimental usando un conjunto de datos obtenidos están disponibles de forma libre: datos de taxi de NYC que consiste de, en promedio 500 mil viajes cada día, sumando más de 868 millones de viajes de taxi en un periodo de cinco años NYC. El tamaño de los datos de entrada, los que están disponibles como archivos *comma-separated values* (CSV), es de 250 GB.

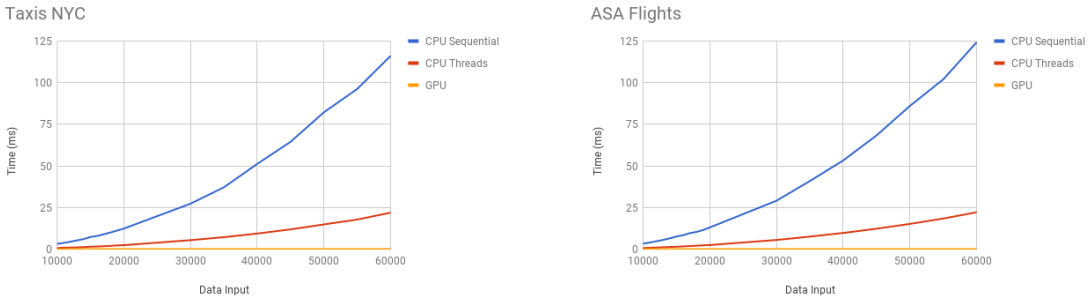
La *New York City Taxi and Limousine Commission* (TLC) colecta información de viajes de taxis en la ciudad, cada viaje consiste de la ubicación y fecha de la partida y llegada junto otros datos relevantes, los que se consideran el número de pasajeros, la distancia del viaje, e información sobre el costo del viaje los que son en total 11 dimensiones.

En la figura 1.2a muestra el crecimiento de tiempo por los datos considerados, tomando en cuenta tiempo lineal, paralelo usando CPU y paralelismo masivo en GPU.

Nuestro siguiente conjunto de datos consta de un historial de vuelos comerciales, el que contiene cada vuelo comercial en el país de Estados Unidos por un periodo de más de 20 años los cuales son recolectados por el departamento de transportes de Estados Unidos con el objetivo de medir el desempeño de los mismos. Los registros incluyen datos respectivos a los vuelos como la fecha y hora del vuelo de partida y como también la de llegada junto con el aeropuerto donde se realizó dicho vuelo; además de incluir información sobre la demora del vuelo, distancia recorrida lo que resulta una cantidad de 13 dimensiones que son consideradas para este experimento. Este conjunto de datos

Taxis NYC			
Entrada de datos	**CPU Secuencial (ms)**	**CPU Threads (ms)**	**GPU (ms)**
10000	3.258	0.665	0
11000	3.953	0.815	0
12000	4.705	0.96	0
13000	5.503	1.102	0
14000	6.469	1.263	0
15000	7.572	1.424	0
16000	8.359	1.645	0
17000	9.581	1.839	0
18000	10.32	2.019	0
19000	11.45	2.275	0
20000	13.009	2.454	0
30000	29.073	5.496	0
35000	40.894	7.51	0
40000	53.252	9.692	0
45000	68.334	12.264	0
50000	86.049	15.144	0
55000	101.985	18.369	0
60000	124.304	22.137	0

Cuadro 1.2: Comparación entre CPU secuencial, CPU paralelizado y procesamiento masivo en GPU. Para el conjunto de datos de taxis de NYC.



(a) Taxis de NYC

(b) Vuelos comerciales en Estados Unidos.

Figura 1.2: Crecimiento del tiempo con respecto al número de elementos, usando el conjunto de datos de taxis de NYC y de vuelos comerciales en Estados Unidos.

cuenta con un total de más de 120 millones de vuelos los cuales están disponibles al público en un periodo de 20 años, de 1978 al 2008.

En el cuadro 1.2 presentamos el tiempo de procesamiento usando este conjunto de datos con los mismos parámetros usados en el cuadro 1.1, el que muestra una tendencia similar que puede ser vista en la figura 1.2b la que va aumentando siguiendo una curva exponencial con respecto al tamaño de datos de entrada.

Una alternativa para reducir los requerimientos de memoria y tiempo en el cálculo de distancia entre los datos, es realizar una proyección multidimensional a un espacio de baja dimensión. Con el fin de ver este comportamiento, ilustramos el desempeño en términos de tiempo y uso de memoria para el proceso de proyectar los conjuntos de datos mostrados en la Tabla 1.4 a 1, 2 y 3 dimensiones.

Utilizando el método t-SNE, en la Figura 1.3 vemos el tiempo requerido en se-

Conjunto de datos	Objetos	Dimensiones
balance	625	4
car	1728	6
solar	1389	10
nursery	12960	8
suicide	27800	12

Cuadro 1.3: Descripción de bases de datos usada para la evaluación de métodos de proyección.

gundos para la proyección por cada conjunto de datos (Maaten y Hinton, 2008). Como se puede observar, el tiempo para proyectar a las dimensiones requeridas es similar utilizando 1.3.

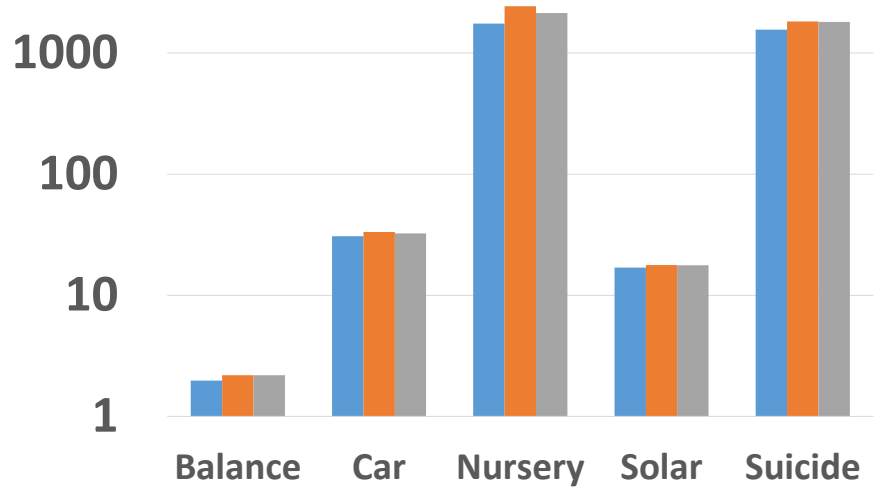


Figura 1.3: Tiempo en segundos tomado para la proyección de t-SNE. 1-D (▬), 2-D (▬), 3-D (▬).

Por el contrario, si el método PCA es usado, la proyección a una dimensión es la que más requiere tiempo para su cálculo (Jolliffe, 2011). Este fenómeno puede ser visto en la Figura 1.4.

Por el lado de la memoria, ambos métodos reducen el espacio requerido bajo el mismo factor directamente proporcional a las dimensiones resultantes. Es decir, el conjunto de datos con menor espacio requerido, es el proyectado a una dimensión.

Sin embargo, una de las cosas que sucede al realizar una reducción de dimensionalidad, es la perdida de presición. En la Figura 1.5 podemos observar este fenomeno despues de evaluar las proyecciones con la métrica de preservación de vecindad (Martins et al., 2015). Es evidente, en ambos casos que a menor es el espacio proyectado, mayor será la precisión perdida.

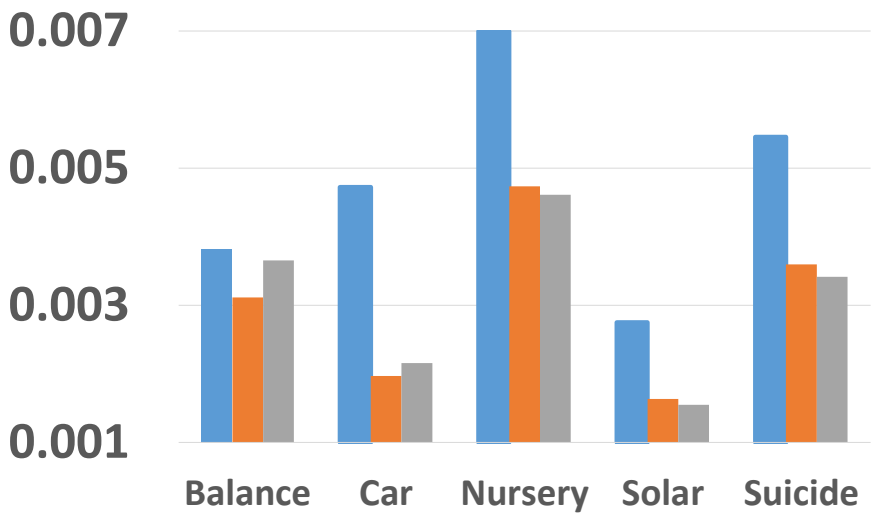


Figura 1.4: Tiempo en segundos tomado para la proyección de PCA. 1-D (▬), 2-D (▬), 3-D (▬).

	Balance	**Car**	**Nursery**	**Solar**	**Suicide**
Original Space	5626	22465	220323	29173	482287
PCA 1-D	4281	16707	70995	13475	15540
PCA 2-D	7781	24483	129315	25793	283070
PCA 3-D	13426	32259	187635	38832	410733
t-SNE 1-D	4898	16356	121844	13078	266716
t-SNE 2-D	11202	31041	230792	24702	505203
t-SNE 3-D	16446	45814	340005	36904	744270

Cuadro 1.4: Espacio en memoria de los conjuntos de datos proyectados a 1, 2 y 3 dimensiones, utilizando PCA y t-SNE

1.2. Planteamiento del problema

Definir una método de indexación a un costo viable que permita la consulta por similitud en tiempo real. Nuestro método se basa en la combinación de métodos de reducción de dimensionalidad con estructuras de datos multidimensionales. Adicionalmente, nuestro método provee una exploración de similitud visual de datos en tiempo real tomando como consulta uno o varios registros simultáneamente en un mapa geográfico.

1.3. Objetivos

Desarrollar una estrategia la indexación de grandes conjuntos de datos georreferenciados para la búsqueda por similitud, visualización y exploración en tiempo real.

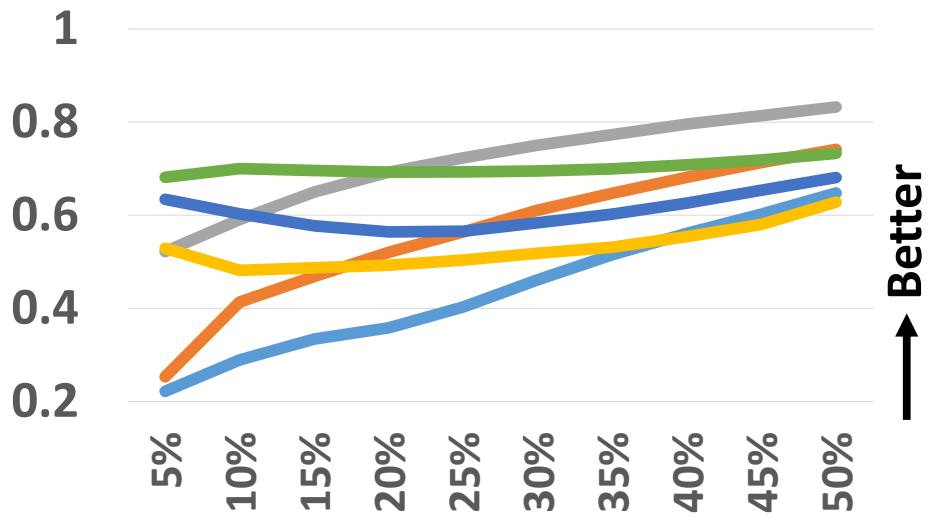


Figura 1.5: Preservación de vecindad de PCA y t-SNE a 1, 2 y 3 dimensiones. PCA 1-D (▬), PCA 2-D (▬), PCA 3-D (▬), t-SNE 1-D (▬), t-SNE 2-D (▬), t-SNE 3-D (▬)

1.3.1. Objetivos específicos

- Proporcionar una exploración visual en tiempo real de datos similares al consultar una o más entradas simultáneamente en un mapa geográfico.
- **Prototipo visual:** Hacer pruebas con casos de estudio que resalten las fortalezas y debilidades en términos de la utilización de espacio, tiempo y latencia.
- **Análisis cualitativo y cuantitativo:** Un conjunto de casos para demostrar la utilidad usando conjunto de datos públicos reales.
- Una discusión sobre el aporte y las características brindadas por la indexación incluyendo sus limitaciones y futuras investigaciones.

1.4. Organización de la tesis

Este trabajo está organizado de la siguiente manera: En el capítulo 2 haremos una categorización de las técnicas existentes en el estado del arte, así como una descripción de cada una de ellas resaltando sus limitaciones y haciendo una comparación de todas ellas, en el capítulo 3 veremos la contribución que realizada por esta investigación, en el capítulo 4 mostraremos nuestros resultados obtenidos realizando pruebas utilizando diferentes bases de datos.

Capítulo 2

Trabajos relacionados

2.1. Consideraciones iniciales

En este capítulo describiremos algunos de los métodos de indexación que abordan el problema de la visualización y exploración de grandes cantidades de datos multidimensionales espacio-temporales, así también exploraremos métodos de mapeamiento de datos multidimensionales a un espacio de una dimensión.

Empezaremos mencionando cuales son las métodos utilizados para la indexación de grandes volúmenes de datos, clasificándolos y agrupándolos por sus características en común. Luego de dar un detalle de cada método, listaremos las ventajas y desventajas presentes. Adicionalmente, también abordaremos algunos métodos utilizados para la proyección a 1 dimensión con el propósito de seleccionar el más adecuado en conjunto a un método de indexación para así poder realizar operaciones de similitud sobre el conjunto de datos.

2.2. Métodos de indexación multidimensional

Con el objetivo de clasificar los métodos de indexación vistos en el estado del arte, hemos dividido estos métodos en tres grandes grupos: basados en *Data Cubes*, acelerados por procesamiento paralelo y sensibles a semántica.

2.2.1. Métodos basados en *Data Cubes*

Muchas bases de datos relacionales abordan este problema implementando el método Data Cubes (Gray et al., 1997), un Data Cube es una estructura la cual realiza agregaciones sobre cada posible conjunto de dimensiones de una tabla en la base de datos, con el fin de lograr una rápida exploración (una agregación representa la idea de

seleccionar un cierto grupo de registros de una tabla y agruparla usando una función de agregación, como conteo, suma, máximo, mínimo, etc.), estos conceptos se pueden ver reflejados en la figura 2.1 que muestra visualmente data cubes con agregaciones de hasta tres dimensiones. El tamaño de un Data Cube es $\prod_i b_i$, donde b_i es el total del resultado de realizar agregaciones en dicha dimensión i. Los *Data Cubes* son frecuentemente problemáticos ya que a medida que el número de dimensiones incrementa puede llegar a necesitar un excesivo tamaño de memoria. Dicho de otro modo, incluso con la mínima representación de un Data cubes el espacio tiende a crecer exponencialmente con la adición de nuevas dimensiones, es decir, no se puede dar una solución escalable a la maldición de la dimensionalidad Bellman et al. (1957).

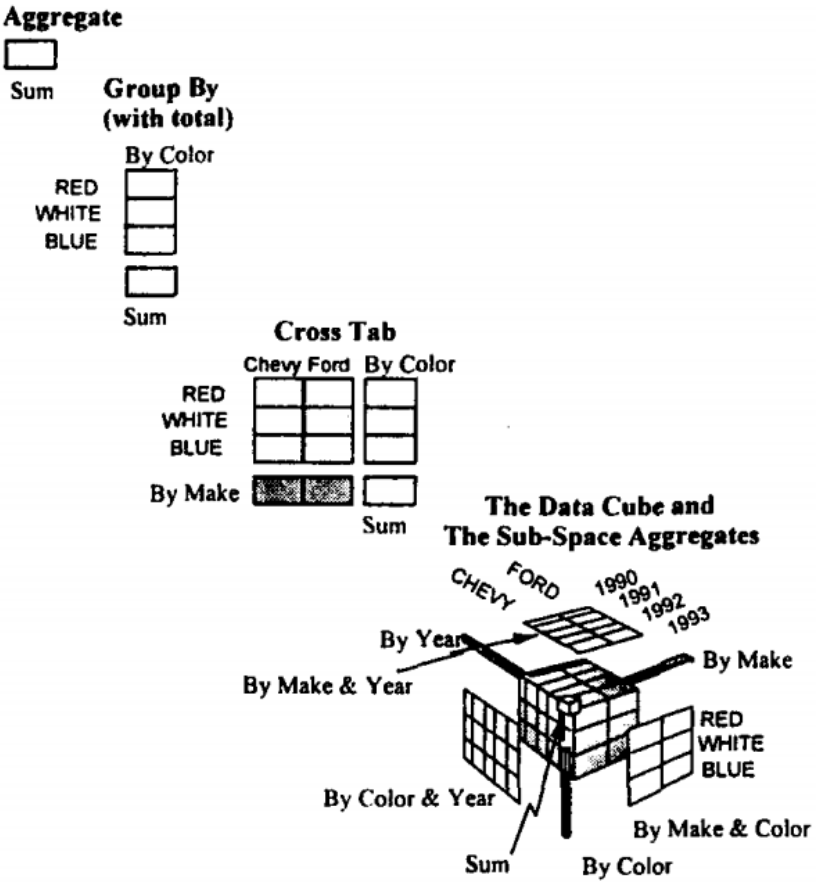


Figura 2.1: Ejemplos de Data Cubes: para 0 dimensiones es un punto, para 1 dimension es una linea con un punto, para 2 dimensiones es una tabla, plano, dos lineas y un punto, para 3 dimensiones es un cubo con 3 tablas intersectadas. Extraido de (Gray et al., 1997)

Las siguientes métodos toman el clásico Data Cube, *On-Line Analytical Processing* (OLAP), para adaptarlo a los requisitos específicos de visualización mostrando la relación fundamental entre Data Cubes, agregaciones y visualización interactiva. Esencialmente cualquier estructura basada en Data Cubes recupera un número de agregaciones de un conjunto base de agregaciones codificándolas en una estructura esparza basada en punteros con el fin de representar o materializar un Data Cube completo eficientemente.

2.2.1.1. Nanocubes

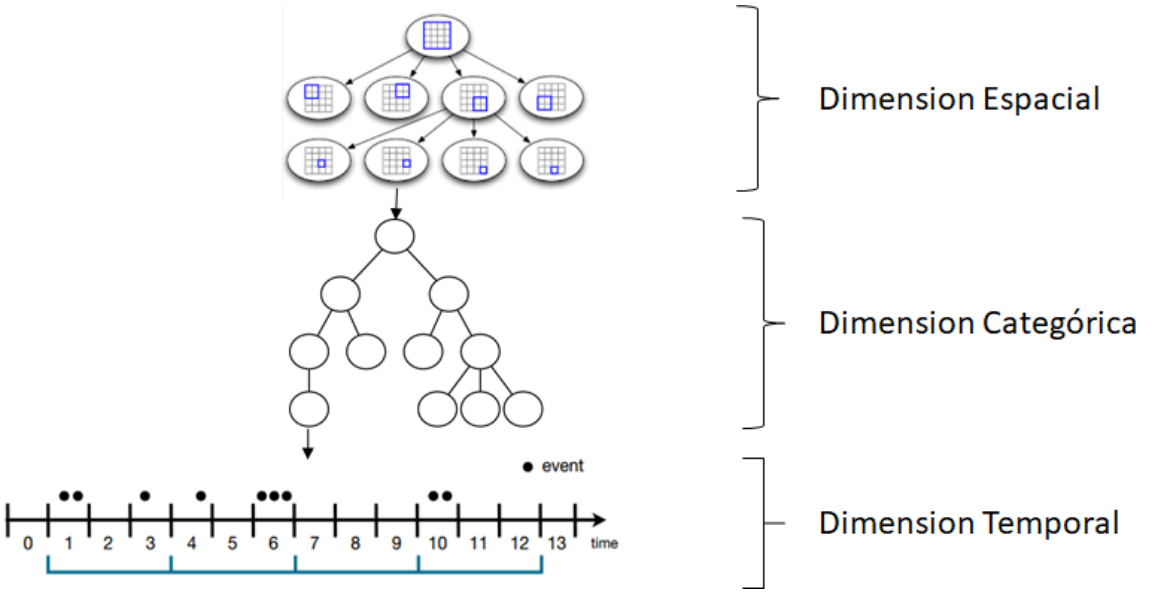


Figura 2.2: Ilustración de la estructura de Nanocubes. Adaptado de (Lins et al., 2013)

Nanocubes (Lins et al., 2013) es una variación compacta de un Data Cube la cual permite la visualización en tiempo real de grandes cantidades de datos multi-dimensionales y espacio-temporales. Nanocubes define para cada nivel diferente de dimensiones independientes una llave de indexación la que luego es usada para combinar agregaciones. La idea principal es encontrar para cada objeto una dirección que define las llaves de indexación para un esquema (conjunto de indexaciones) y actualizar el Nanocubes en la inserción.

En la construcción de un Nanocubes se va maximizando la cantidad de links entre dimensiones con el fin de permitir la agregación de consultas de un conjunto pequeño. Las dimensiones son agrupadas según su tipo y son recorridas siempre bajo un orden determinado: espacial, categórico y temporal.

Para la representación de la dimensión espacial hace uso de un Quadtree donde un mapa es dividido en pequeños cuadrantes. Cada nodo del Quadtree contiene un árbol plano (dado que solo tiene un nivel de profundidad) la que es utilizada para representar las dimensiones categóricas, consiste en un nodo raíz con tantos hijos como diferentes categorías asociadas al padre. Finalmente, el contenido de los nodos hijos del árbol plano son series de tiempo de los puntos que cumplen las restricciones espaciales del Quadtree y categóricas por el árbol plano. Las series de tiempo son almacenadas como una variante esparza de una tabla acumulativa. Un ejemplo del resultado de un Nanocubes se puede observar en la Figura 2.2.

La ventaja de la representación con Nanocubes, es que un nodo padre siempre es un resumen del contenido de sus nodos hijos, por lo que permite consultas rápidas a la estructura de datos ya que existe series de tiempo pre-calculadas para cada resolución en dimensiones espaciales y categóricas.

En la inserción de datos existe una saturación de llaves, es decir, existe un rápido crecimiento inicial de memoria para luego estabilizarse ya que la mayoría de los objetos ya insertados no requieren más memoria dado que las llaves ya fueron almacenadas.

Cuando el espacio ocupado por el Nanocubes es grande, la lógica y memoria extra necesaria para manejar los punteros llega a ser excesivo por el espacio requerido de las llaves debido a los objetos insertados como ocurre con cualquier estructura de datos esparza. La estructura de datos almacena el resultado para cada agregación por lo que requiere que la función de agregación un pre-requisito para la construcción del Nanocubes.

Como resultado de una consulta al Nanocubes se obtiene agregaciones a través de una o más dimensiones, por lo que no permite recuperar registros individuales como una base de datos tradicional ya que no se almacena los registros originales de los datos (solo acumulaciones). Aunque la complejidad total de espacio requerido por Nanocubes puede ser almacenada en la memoria principal de una laptop moderna tiende a ser exponencial al tamaño del conjunto de indexación.

En la implementación del Nanocubes, regiones arbitrarias no son soportadas por el Quadtree y la construcción no fue optimizada en términos de velocidad (paralelismo o *memory pools*[1]), además de solo permitir agregar objetos. Así mismo Nanocubes solo permite una dimensión espacial y una temporal.

2.2.1.2. Hashedcubes

Hashedcubes (Pahins et al., 2017) es una alternativa a Nanocubes, la principal diferencia es encuentra en evitar realizar un gran número de agregaciones usando un esquema de ordenamiento parcial junto con pivotes permitiendo rápidas consultas y una implementación compacta y más simple. Para esto Hashedcubes sigue las siguientes consideraciones: al ordenar un arreglo su tamaño no es alterado, podemos representar un sub-arreglo contiguo mediante pivotes desde el inicio al fin del mismo, y finalmente este sub-arreglo puede tener un orden interno.

El algoritmo inicia con una representación de los datos con un pivote que cubre todo el arreglo, luego y sucesivamente permuta este arreglo ordenandolo según una dimensión e ir almacenando los pivotes que delimitan los valores de dicho arreglo. Esta representación permite rápidamente evitar ejecuciones al arreglo total de datos.

Este algoritmo requiere un ordenamiento para las dimensiones del conjunto de datos, aunque Hashedcubes puede procesar las dimensiones en cualquier orden los autores eligieron usar la dimensión espacial primero, con el fin de incrementar la velocidad en la que consultas geo-espaciales pueden ser resueltas. Es importante notar que intercambiar el orden en el cual las variables son ordenadas impactan tanto en el uso de memoria como en el tiempo de ejecución.

[1]Pre-asignación de un número de bloques de memoria.

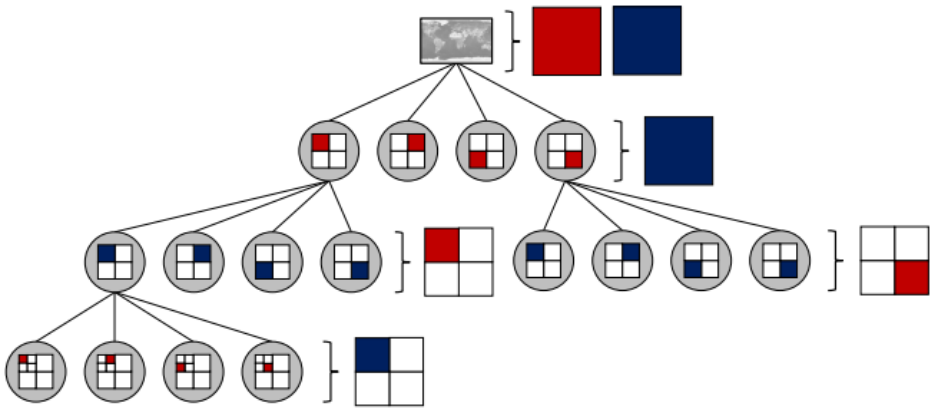


Figura 2.3: Ejemplo de un quadtree para dos dimensiones espaciales. Extraido de (Wang et al., 2017)

Hashedcubes no pre-calcula agregaciones sobre cada posible conjunto de dimensiones, en cambio, calcula las agregaciones faltantes gracias a su jerarquía de pivotes, por lo que una consulta probablemente requerirá un barrido sobre una gran porción del arreglo. La mayoría de las consultas que consumían mucho tiempo requerían gran número de agregaciones de muchos pivotes pequeños, teniendo en el peor caso un peor desempeño que otras estructuras de datos, pero permitiendo un menor consumo de memoria.

Otra de las ventajas de Hashedcubes es que soporta esquemas de múltiples dimensiones espaciales atravesando una única estructura de datos entrelazando la construcción de cada Quadtree, este efecto es ilustrado en la Figura 2.3.

El uso de la memoria es directamente proporcional a la cantidad de pivotes por lo que también presenta el fenómeno de saturación de llaves debido a que la mayoría de los registros ya insertados no requieren memoria adicional, dado que los pivotes ya fueron indexados. Otra de las ventajas brindadas por los pivotes es la recuperación de la data original mediante la asociación de pivotes indexados con un índice de una base de datos externa.

Al contrario del trabajo anteriormente mencionado, Nanocubes, el tiempo de construcción del Hashedcubes ha sido optimizado evitando (de)asignaciones de memoria. En su implementación Hashedcubes es fácilmente paralelizable debido a que la estructura ya no es alterada después de ser construida.

Además, para balancear el tiempo de ejecución, el uso de memoria y precisión de la visualización adoptan un mínimo tamaño para las hojas del Quadtree creando pivotes truncados.

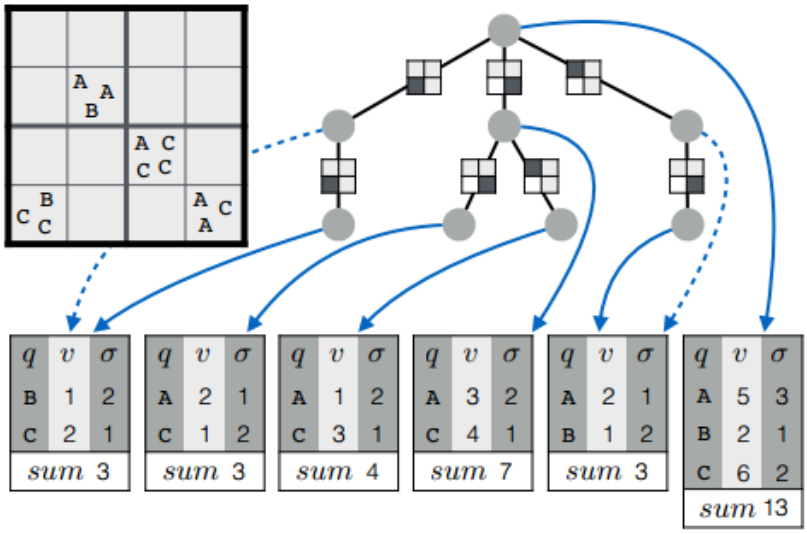


Figura 2.4: Ejemplo de un TOPKUBE con una dimension espacial y una dimensión llave para el conteo y ranking de dos tipos de eventos: A, B o C. Extraido de (Miranda et al., 2017)

2.2.1.3. Gaussian Cubes

Los trabajos anteriormente mencionados producen visualizaciones interactivas a través de resúmenes visuales, pero están limitados a una clase de análisis, por ejemplo, no soportan modelos estadísticos. Gaussian Cubes (Wang et al., 2017) en vez de pre-calcular agregaciones de sub-conjuntos de los datos, pre-calcula la mejor distribución gaussiana multivariante para un conjunto dado de variables y así poder trabajar con millones de datos de forma interactiva.

Gaussian Cubes está implementado como una extensión de Nanocubes, por lo que hereda su rendimiento en tiempo de ejecución, requerimiento de memoria, sintaxis y velocidad.

El sistema interactivo con capacidad de modelado ofrecido por Gaussian Cubes incluye, pero no está limitado a mínimos cuadrados y análisis de componente principales, pero dichos modelos se encuentran determinados por los valores pre-agregados.

Gaussian Cubes sufre de espacio adicional que es cuadrático al tamaño de las variables a modelar, por esta razón si el usuario quiere explorar el conjunto de datos para cualquier dimensión se requerirá mucha más memoria.

Otra de las limitaciones del modelado de Gaussian Cubes es la ausencia de una evaluación del modelo, esto es cuan bien el modelo es ajustado, para brindar la capacidad de diagnosticar la ejecución al usuario.

2.2.1.4. TOPKUBE

TOPKUBE (Miranda et al., 2017) es una extensión al esquema de pre-agregaciones descrito en Nanocubes, limitada solo a un tipo de análisis para cubrir un aspecto importante que no ha sido aún abordado: Top ranking de objetos de una gran colección de datos multi-dimensionales a partir de una selección arbitraria. Identificar los *top-k* objetos sin una información del ranking codificada es buscar para cada objeto su valor de medida asociado y reportar los *top-k* encontrados cuyo cálculo es proporcional a los objetos en la selección de interés.

TOPKUBE incluye información de ranking para acelerar consultas del tipo *top-k* sobre una de las dimensiones codificándolo en una nueva dimensión llamándola dimensión llave la que contiene agregaciones de sus dimensiones predecesoras, esto se puede ver reflejado en la figura 2.4 donde las hojas almacenan una tabla para calcular el ranking de cada objeto. Ya que el número de agregaciones requeridas para responder una selección multidimensional es más que una, se da solución a este problema nombrado como "transformación de listas rankeadas a *top-k*".

El tamaño de un TOPKUBE es igual al Nanocubes más la codificación extra de arreglos para las consultas *top-k*. Para la representación de la dimensión temporal, TOPKUBE utiliza un árbol binario, por lo que añade un costo multiplicativo logarítmico al tamaño de dicha dimensión con la ventaja de soportar múltiples dimensiones temporales.

La elección del umbral para el problema de la "transformación de listas rankeadas a *top-k*" tiene un gran impacto en la velocidad. En la implementación del TOPKUBE la construcción y utilización de memoria no han sido optimizadas, además de no utilizar paralelismo en CPU.

2.2.2. Métodos acelerados por procesamiento paralelo

Estos métodos tienen en común el uso de paralelismo tanto por CPU como GPU, o ambos. El objetivo es partir el procesamiento en sub-tareas que son llevadas a cabo secuencialmente con el fin de ayudar a superar algún cuello de botella existente en el proceso tanto de construcción del índice como de las consultas sobre los datos.

2.2.2.1. *Spatio-Temporal Indexing using GPUs* (STIG)

STIG (Doraiswamy et al., 2016) es un esquema de indexación que hace uso de GPU para manejar consultas espacio-temporales. Utiliza almacenamiento basado en bloques sobre datos históricos.

La estrategia usada es de un filtro simultáno sobre múltiples dimensiones haciendo uso efectivo del uso de GPU para realizar rápidas operaciones sobre numerosas pruebas

independientes entre sí. STIG es una generalización de la estructura KD-tree que es usado para aprovechar el uso del GPU. También consta de diferentes estrategias para la ejecución de consultas para soportar diferentes configuraciones incluyendo: Solo GPU, solo CPU y una combinación híbrida de CPU y GPU. Estas estrategias pueden ser usadas para aprovechar múltiples GPU en la nueva característica presentada por las nuevas tarjetas NVIDIA de procesadores gráficos, paralelismo dinámico.

STIG consiste de dos componentes: KD-tree y un conjunto de bloques para almacenar registros. Cada nodo en la estructura KD-tree apunta a un bloque el cual almacena los datos correspondientes a una colección de registros. Bloques hojas candidatos son enviados al GPU donde son buscados en paralelo. El agrupamiento de puntos minimiza la cantidad de datos a ser transferidos entre CPU y GPU, en general permite el procesamiento balanceado de todos los GPUs.

Para mantener estas propiedades el árbol debe mantenerse equilibrado y para hacer uso efecto del GPU los componentes del índice deben mantenerse en memoria contigua otra desventaja es el tipo de datos que soporta la implementación del STIG, limitándose solamente a datos de los que se puedan obtener una mediana. En la implementación del STIG requiere que el usuario re-cree el índice periódicamente, otra principal limitación es que está implementada en *Compute Unified Device Architecture* (CUDA) por ende solo funciona con tarjetas NVIDIA.

2.2.3. Métodos sensibles a semántica

En esta sección se detalla los índices que obtienen el contenido conceptual del conjunto de datos y establecen una relación entre términos con contexto similar.

2.2.3.1. SemTree

Para dar solución a la indexación de documentos, Amato et al. (2015) desarrollaron una versión distribuida de la estructura KD-tree.

Semtree es un indice distribuido adecuado para el manejo de datos semanticos que soporta la recuperación de información de una gran cantidad de colecciones de documentos, la semántica de cada documento es representada por un conjunto de tres atributos, <sujeto, predicado, objeto>, se define una distancia semántica entre dos tripletes de la siguiente forma:

$$d(t_i, t_j) = \alpha * d_s(t_i^s, t_j^s) + \beta * d_p(t_i^p, t_j^p) + \lambda * d_o(t_i^o, t_j^o) \tag{2.1}$$

Donde t_k^s, t_k^p y t_k^o son la proyección de un triplete t^k para el sujeto, predicado y objeto respectivamente, $d_s(t_i^s, t_j^s)$, $d_p(t_i^p, t_j^p)$ y $d_o(t_i^o, t_j^o)$ son las distancias entre los atributos de sujeto, predicado y objeto respectivamente y α, β y λ son los pesos de

forma que $\alpha + \beta + \lambda = 1$. Para luego mapearlos a un espacio vectorial usando el algoritmo *FastMap* (Faloutsos y Lin, 1995) en el cual es posible definir una estructura de indexación para trabajar en un ambiente distribuido.

Al insertar un objeto el algoritmo inicia de la raíz a las hojas, encontrando la posición adecuada para ser insertado. Si alguna hoja sobrecarga la partición en la que se encuentra, dos nodos son creados y ubicados en una nueva partición. Una ilustración de este proceso se puede observar en la Figura 2.5

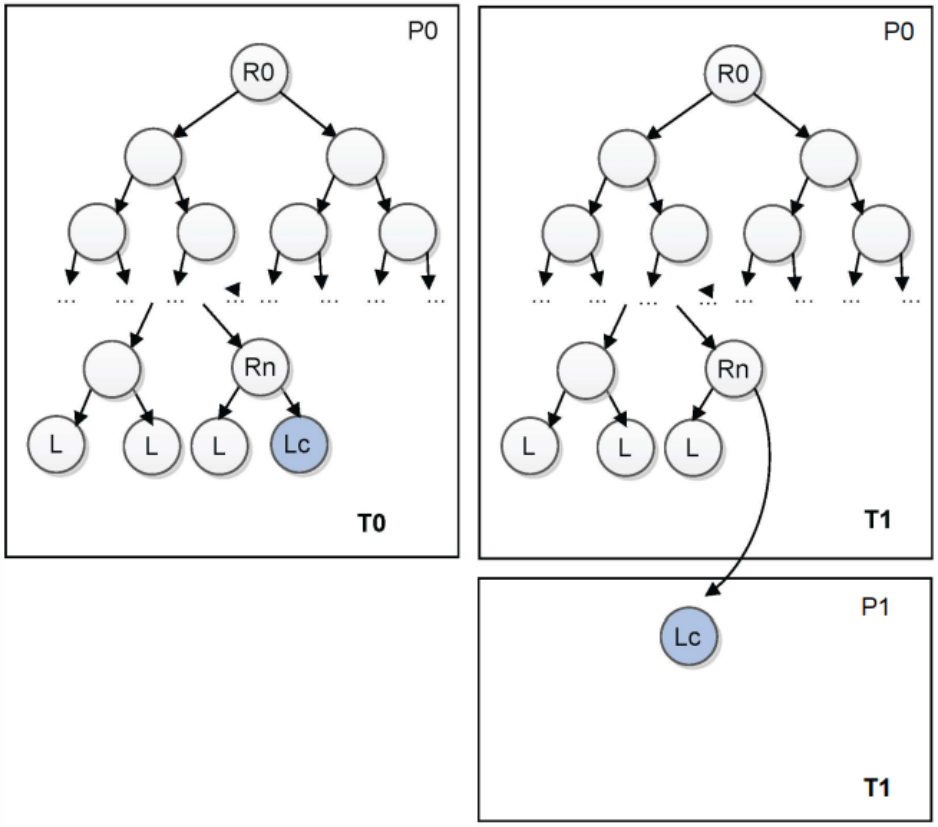


Figura 2.5: Ilustración de la estrategia del SemTree. Adaptado de (Amato et al., 2015)

Como resultado de esta estrategia de partición adoptada por el Semtree, ciertas particiones tienden a contener nodos únicamente para ubicar otras particiones, y por otro lado algunas particiones solo contienen nodos con los objetos insertados.

2.2.3.2. Grafo de vecindad

Para encontrar los AKNN sobre descriptores visuales se ha propuesto una partición aleatoria junto con un diseño de dos capas para la indexación de vecindades (Zhou et al., 2013).

La idea principal es dividir el espacio total de objetos eligiendo un número de pivotes aleatorio e ir agrupando los AKNN a cada pivote, luego y recursivamente remover los pivotes cuyo número de elementos agrupados sobrepasen los límites inferior

y superior e ir reemplazandolos por nuevos pivotes. Como resultado se puede distribuir los grupos por diferentes máquinas para su búsqueda como se puede observar en la Figura 2.6.

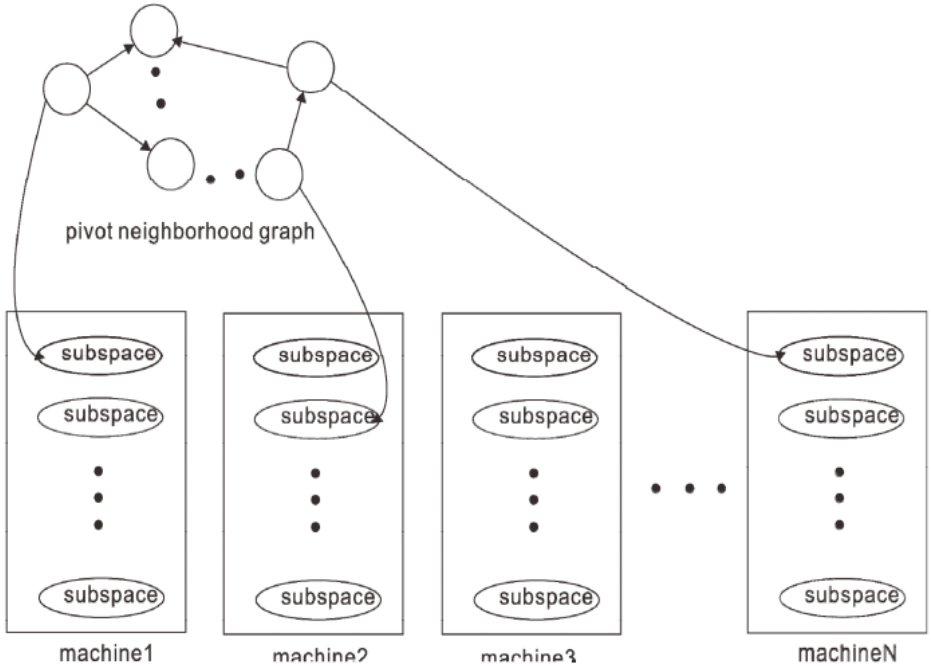


Figura 2.6: Ejemplo de un grafo de vecindad. Adaptado de (Zhou et al., 2013)

Cada nodo máquina almacenará el pivote junto con los datos y la topología en conjunto de todos los datos distribuidos. Al realizar una búsqueda de los AKNN solo será necesario consultar por los pivotes KNN para luego recuperar los AKNN de la mezcla del resultado previo.

2.2.4. Analizando todos los métodos

Todos los métodos descritos en este capítulo son resumidos en el Cuadro 2.1. Se puede evidenciar inicialmente que HashedCubes posee las mayores características debido a su simplicidad y la posibilidad de poder representar muchos atributos espaciales, pero a costo de consultas con una mayor latencia. El método TOPKUBES por el contrario, mantiene las limitaciones que presenta NanoCubes, es decir, no posee la capacidad de soportar muchas dimensiones espaciales, no puede se puede actualizar si un registro es eliminado, etc. De forma similar Gaussian Cubes presenta las mismas características de NanoCubes, pero permite otro tipo de operaciones: modelado estadísticos del conjunto de datos.

Como alternativa a la familia de métodos basados en la estructura Data Cubes tenemos a STIG que da soporte a muchas dimensiones espaciales y temporales. STIG gracias a su procesamiento paralelo es el que tiene mejor rendimiento de tiempo para la recuperación de datos entre todos los métodos descritos. Pero está limitado a la

indexación de datos que permitan la obtención de la mediana.

Por otro lado, para métodos de tipo semántico, SemTree aborda este problema enfocándose únicamente en documentos. Finalmente, el Grafo de Vecindad discutido nos permite la recuperación de datos cercanos a un punto en una gran cantidad volumen de datos de alta dimensionalidad más no en datos geo-referenciados.

2.3. Métodos de mapeamiento de datos al espacio dimensional $1D$

En esta sección, discutiremos tres metodos para la reducción a una dimensón: Vector de Fiedler, *Z-Order curve* y *Hilbert Curve*.

Un mapeamiento de dimensionalidad es usualmente usado como pre-procesamiento el cual tiene el objetivo de reducir a un número las dimensiones bajo cierto criterio. Estos métodos mapean el conjunto de datos a subespacios derivados del espacio original, de menor dimensión, que permiten hacer una descripción de los datos a un menor costo.

2.3.1. Vector de Fiedler

El vector de Fiedler es obtenido calculando el autovector correspondiente al segundo menor autovalor no nulo de la matriz Laplaciana. (Chung, 1996) El cual es aplicado sobre un grafo G que representa nuestro conjunto de datos.

Es decir, para un grafo $G = (V, A)$ con N vértices, los autovalores de la matriz Laplaciana son definidos como sigue:

$$0 = \mu_0 \leq \mu_1 \leq ...\mu_N \tag{2.2}$$

Donde sus correspondientes autovectores pueden ser escritos de la siguiente forma: $v_0, v_1, ...v_N$

El vector de Fiedler es el resultado de v_0 correspondiente al segundo menor autovalor (μ_1). La magnitud de este autovalor refleja la conectividad del grafo G. Por ejemplo, el valor es mayor a cero si y sólo si G es un grafo conexo, esto se deriva del hecho que el número de veces que cero aparece como un autovalor en la matriz Laplaciana es el número de componentes conectados en el gago G. Del mismo modo a medida que el valor es pequeño el grafo G posee una estructura más modular comparado a grafos con un valor más elevado.

Para obtener el grafo G correspondiente a nuestro conjunto de datos, conectamos

	Métodos de indexación							
Características	**ImMens**	**NanoCubes**	**HashedCubes**	**Gaussian Cubes**	**STIG**	**TOPKUBES**	**SemTree**	**Grafo de Vecindad**
Multi-Espacial	✓	✗	✓	✗	✓	✗	✗	✓
Multi-Temporal	✓	✗	✓	✗	✓	✓	✗	✓
Categóricos	✓	✓	✓	✓	✗	✓	✗	✗
Incremental	✗	✗	✗	✗	✗	✗	✓	✗
Saturatión de Llaves	✗	✓	✓	✓	✗	✓	✗	✗
Paralelismo	✓	✗	✗	✗	✓	✗	✓	✗
Repetidos *mem (de)alloc*	✗	✗	✓	✗	-	✓	-	-
Recuperación de Datos	✗	✗	✓	✗	✗	✗	✓	✓
Pivotes Truncados	✗	✗	✓	✗	✗	✗	✗	✗
Código Fuente	✓	✓	✓	✗	✓	✗	✗	✗

Cuadro 2.1: Resumen de las diferencias de las estructuras de datos mostradas.

cada registro con sus vecinos más cercanos dado un umbral, es importante notar que el umbral debe ser lo suficientemente grande para obtener un grafo conexo. Nuestro siguiente paso es calcular la matriz Laplaciana,definida de la siguiente forma:

$$L = D - A \tag{2.3}$$

donde D es la matriz con el grado de cada vértice del grafo y A es la matriz de adyacencia. El vector resultante representan a los datos proyectados a un espacio de una dimensión.

2.3.2. *Z-Order Curve*

El siguiente método considerado es *Z-Order Curve* (Morton, 1966), una de sus ventajas sobre otros métodos es la simplicidad de su cálculo.

Se empieza ordenando, de menor mayor, todos los valores por cada una de las N dimensiones. En la figura 2.7 se observa un ejemplo de una curva de dos dimensiones con sus respectivos valores ordenados de menor a mayor, de izquierda a derecha (*winter*, *spring*, *summer*, *autumn*) y de arriba hacia abajo (*low*, *medium*, *high*).

Se toma como primer menor valor proyectado cero la instancia con los menores valores de cada dimensión, en nuestro ejemplo 2.7 la casilla superior izquierda (*winter*, *low*), el siguientes valores se obtiene aumentado en uno a todas las dimensiones, en nuestro ejemplo (*spring*, *low*) y (*winter*, *medium*). Este recorrido conecta los valores recursivamente en una curva Z.

El valor proyectado de un punto en N-dimensión puede ser obtenido mediante el cambio de bits. Este proceso se puede observar en la Figura 2.7 donde se muestra un ejemplo de una curva de (*Autumn*, *low*) utilizando el proceso de cambio de bits.

2.3.3. *Hilbert Curve*

Finalmente, nuestro último método es *Hilbert Curve*un sistema de Lindenmayer inventado por Hilbert. (Hubert, 1891)

Hilbert curve es una alternativa al *Z-Order Curve* donde su principal ventaja consiste en nunca realizar saltos largos al describir su curva, esta curva atraviesa los vértices de un poliedro de un hipercubo de N dimensiones en un orden *Gray code* el cual produce un generador para la *Hilbert curve* (Gilbert, 1958), una comparación visual se puede ver en la Figura 2.8.

En la Figura 2.9 mostramos a *Hilbert curve* para el mapeo entre 1 y 2 dimensiones utilizando el mismo ejemplo que el método anterior (*Z-order curve*).

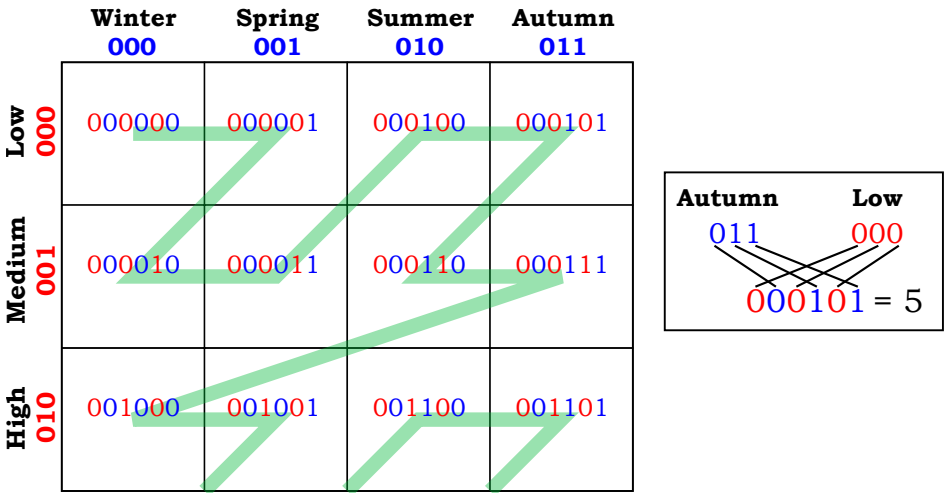


Figura 2.7: Ejemplo de *Z-curve*n.

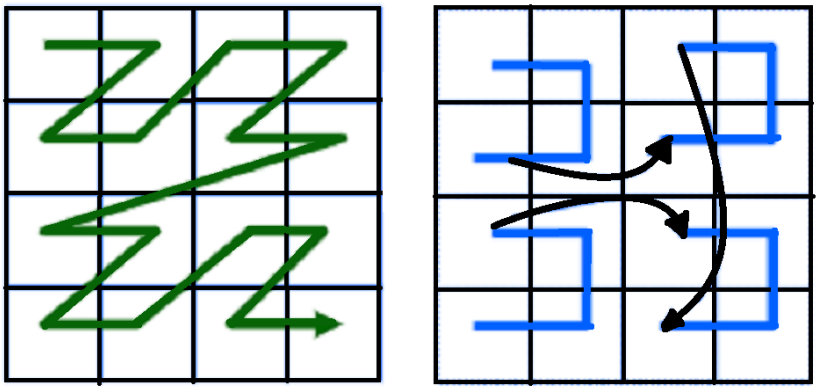


Figura 2.8: Comparación entre *Z-order Curve* y *Gray code*.

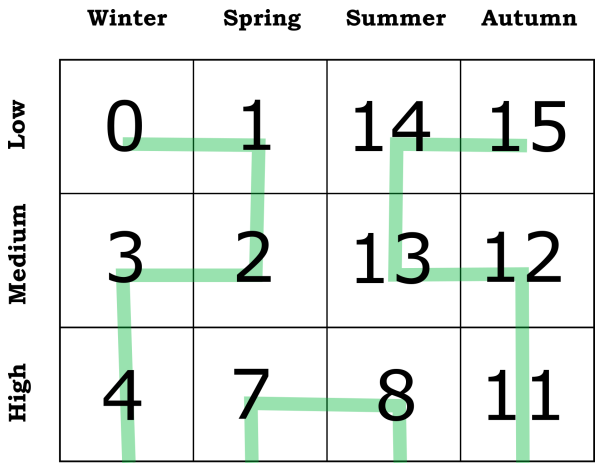


Figura 2.9: *Hilbert curve*, ejemplo de segundo orden.

2.3.4. Analizando todos los métodos

Una de las más grandes desventajas de *Fiedler vector* es su requerimiento de memoria y procesamiento al generar una matriz de un con grandes conjuntos de datos, lo que hace difícil su respuesta para un sistema en tiempo real. Además dar como resultado de su mapeamiento a $1D$ valores continuos. Por el contrario, el rango de los métodos *Z-order curve* y *Hilbert curve* nos brindan valores discretos. Por un lado, obtener el valor *Z-order* es un proceso constante mientras que *Hilbert curve* además de un cambio de bits realiza un post-procesamiento para realizar las rotaciones requeridas.

2.4. Consideraciones finales

En este capítulo fueron presentadas diferentes técnicas tanto para la indexación de datos multidimensionales como para la reducción dimensional a un espacio $1D$. Agrupamos los métodos vistos en el estado del arte por sus diferentes características, resaltando entre ellas: basados en *Data Cubes*, acelerados por procesamiento paralelo y sensibles a semántica. Resumimos las características más importantes en el Cuadro 2.1 para una mejor apreciación de sus ventajas y desventajas.

Capítulo 3

Una estructura de datos para consultas por similitud en grandes volumenes de datos

3.1. Consideraciones iniciales

En este trabajo, teniendo un gran volumen de datos espacio-temporales y multidimensionales; estamos interesados en obtener los elementos de este conjunto que se encuentran cerca a una consulta bajo cierto criterio de similitud. Debido al uso de N-dimensiones, la complejidad de procesar la distancia entre sus elementos es proporcional al número N.

La idea principal es transformar los elementos de un espacio N-dimensional a un espacio de una dimensión utilizando una técnica de reducción multidimensional. Esto con el fin de aprovechar la simplicidad de realizar métodos de búsqueda de rango sobre una dimensión.

A continuación, explicaremos cómo construir y consultar nuestro método. Explicaremos por qué utilizamos en tanto un método de proyección junto con una estructura de datos y por qué funcionan bien juntas. Un resumen de los pasos realizados para la indexación se muestra en la Figura 3.1. Primero, tratamos la complejidad de trabajar en un espacio dimensional elevado utilizando un método de reducción multidimensional para aliviar el cálculo de distancia entre dos elementos. Finalmente, indizar el resultado del mapeo a $1D$ a nuestra estructura de datos para su consulta, como resultado de este proceso mejoramos la latencia de consulta y el requerimiento de memoria.

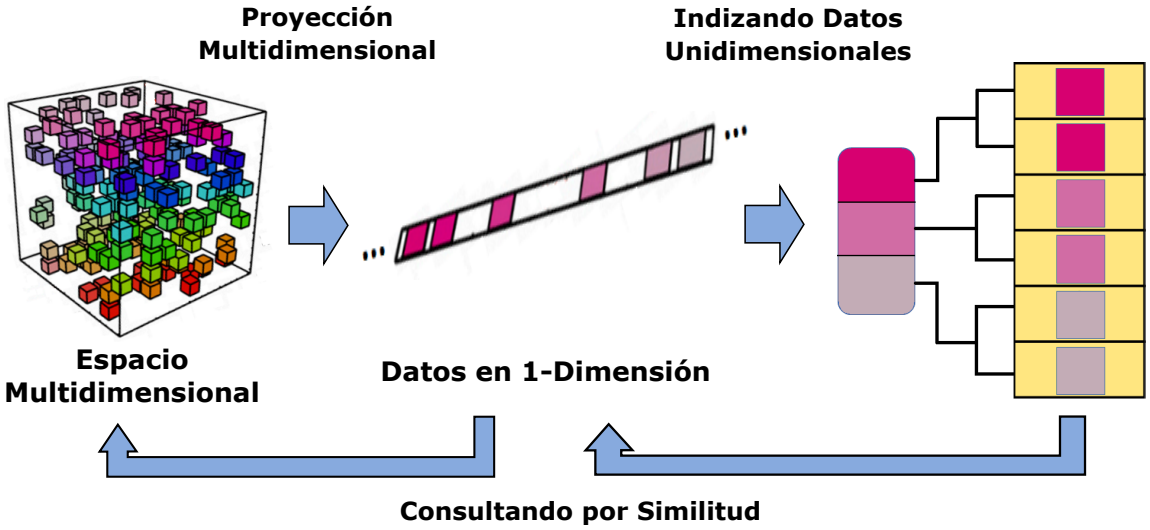


Figura 3.1: Ilustración de los pasos realizados para obtener un orden por similitud entre datos geo-referencidos.

3.2. Proyectando datos a 1D

Para determinar cuál de los métodos descritos anteriormente funciona mejor cuando se preserva la similitud, recopilamos cuatro conjuntos de datos para ser probados por nuestra medida métrica de preservación de vecindad (Martins et al., 2015).

Cada conjunto de datos contiene una matriz de números que varían en tamaño hasta 13000 entradas y valores específicos de dominio con hasta seis valores distintos. Resumimos todas las variaciones de esquema y los conjuntos de datos en la Tabla 1.4.

El conjunto de datos de balance contiene 625 entradas totales con cuatro dimensiones, cada una de las cuales tiene cinco valores diferentes. Cada punto posible se crea una instancia y aparece solo una vez en el conjunto de datos, por lo que se describen completamente en ambas curvas de proyección; *Z-order* y *Hilbert*. En este caso, la proyección de la curva de Hilbert tiene la mejor precisión debido a que se minimizan todos los saltos que podrían haberse producido en el conjunto de datos, Figura 3.2a.

Un caso similar ocurre en el conjunto de datos *car*, que contiene 1728 entradas con una estructura cuadrada que tiene seis dimensiones de un rango entre tres y cuatro valores distintos. Como podemos ver en la Figure 3.2b, la proyección de la curva de Hilbert también obtiene la mejor precisión debido a la estructura cuadrada casi completa en el conjunto de datos incluido anteriormente.

Sin embargo, nuestro próximo conjunto de datos *solar* tiene una definición más irregular porque contiene un total de 1389 entradas, diez dimensiones y un rango de valores desequilibrado de dos a seis variaciones. Podemos observar en la Figura 3.2c que la curva de orden Z adquiere el mejor rendimiento, como resultado de no tener los saltos largos que la curva de Hilbert es sobresaliente.

En nuestro dos últimos ejemplos, los conjuntos de datos de *nursery* y *suicide* (Figura 3.2d, 3.2e), la curva *Z-order* mantiene mejor la relación del conjunto de datos, esto se explica por el hecho de que el conjunto de datos de 12960 y 27800 respectivamente se describen mejor mediante una curva Z ya que sus valores posibles son tienen una variación descrita en forma cuadrangular.

3.3. Construyendo el índice

En esta etapa, los datos ya deben proyectarse en una dimensión en nuestro paso de preprocesamiento. Para realizar esta tarea, podríamos usar la proyección *Z-Order* debido a sus destacados resultados, además de aprovechar el mapeamento directo a $1D$, sin agregar complejidad adicional a nuestro proceso 3.1. Tiendo como resultado valores discretos, la función de agregación se puede usar fácilmente para juntar valores similares. Con esta nueva representación de datos en una dimensión, podemos tomar el concepto de HashedCubes y usarlo para indexar en una única matriz de hash debido a su simplicidad y beneficios sobre los otros métodos.

Nos interesa saber dónde se ubican los elementos más similares en un mapa geográfico con respecto a un grupo de puntos, siguiendo la idea de HashedCubes, primero indexaremos los datos según los atributos espaciales. Seguimos utilizando la estructura de datos de Quadtree, por lo tanto, para cada nodo en el arbol, obtendremos un par de pivotes que delimitan los datos.

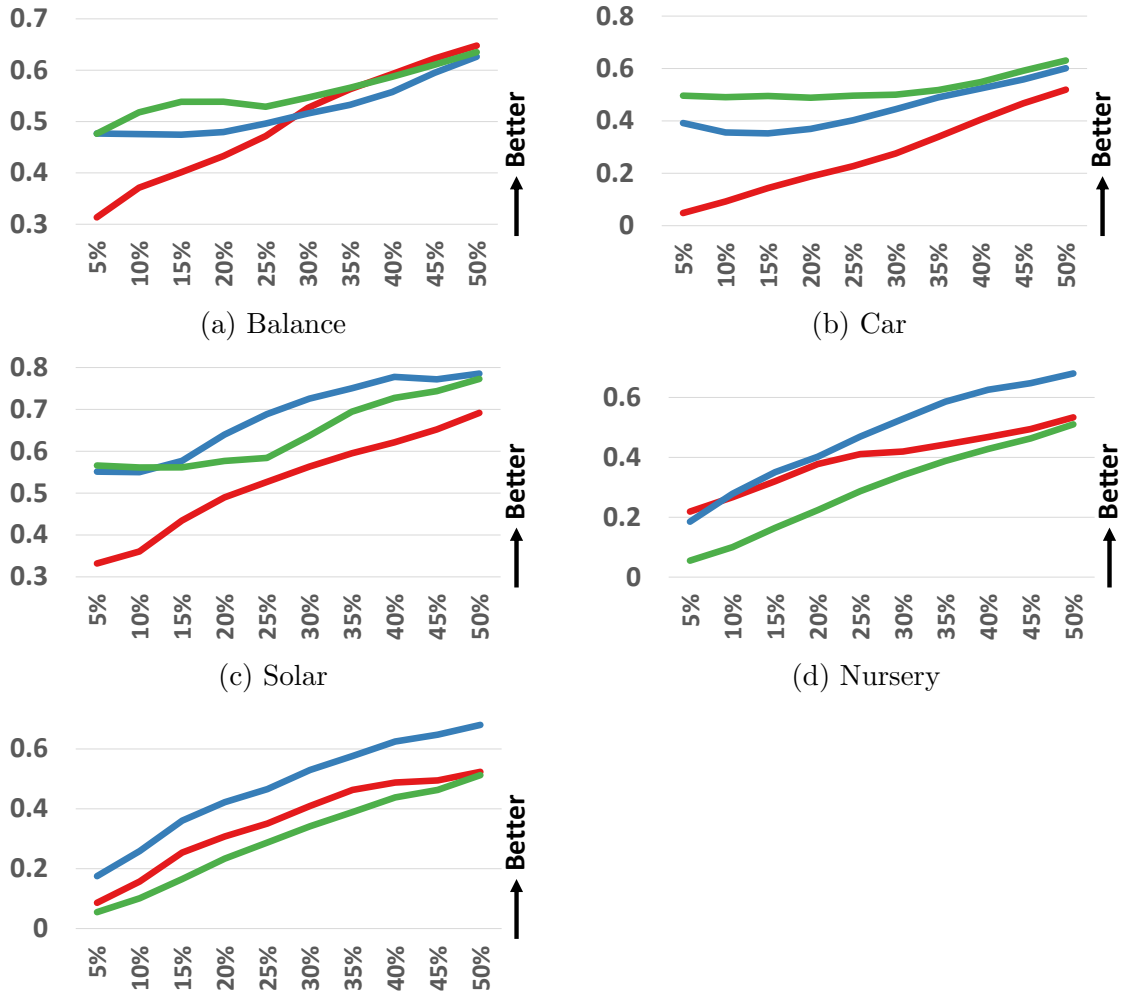
El objetivo es dividir la ubicación de los elementos más similares a una determinada consulta. Por esa razón, por cada nodo en nuestro Quadtree ordenaremos los datos usando su valor de similitud, es decir, el mapeamiento en el espacio $1D$ brindado por el *Z-order curve* descrito anteriormente. Ilustramos cómo construimos nuestra estructura de datos en la Figura 3.3.

Es importante tener en cuenta que, además, solo mostramos cómo indexar por posición geográfica y similitud, nuestro método también admite atributos categóricos y temporales como su antecesor.

3.4. Consultando los datos

Al consultar un HasehdCubes se recorre la estructura atravesando desde la raíz hasta el área la cual queremos consultar. Eso con el fin de obtener solo los pivotes cuyos valores cumplen cierto criterio de rango. La consulta de similitud más natural para un área específica ocurre cuando se consulta un único punto. Sin embargo, en el caso de puntos múltiples, proponemos tres enfoques: Unión, Intersección y Valor medio, (Figura 3.4).

La primera y sencilla es devolver todos los elementos que están cerca de los puntos



(a) Balance (b) Car

(c) Solar (d) Nursery

(e) Suicide

Figura 3.2: Cuantificando la preservación de vecindad para nuestras cuatro bases de datos utilizando *Fiedler Vector* (▬), *Z order curve* (▬) and *Hilbert curve* (▬), el resultado de esta medida es impactado por la distribución de los datos a lo largo de las dimensiones. El mejor metodo a ser usado depende de cuan bien los datos han sido descritos por el *Z-order curve* o por la cantidad de veces los saltos fueron evitados por *Hilbert curve*.

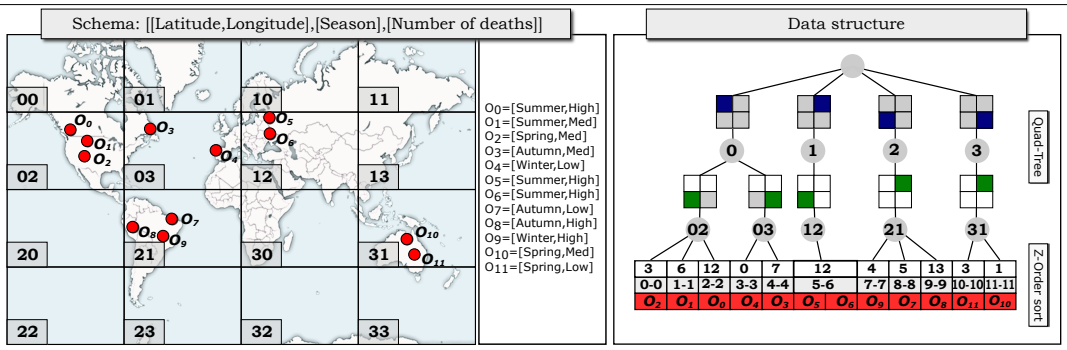
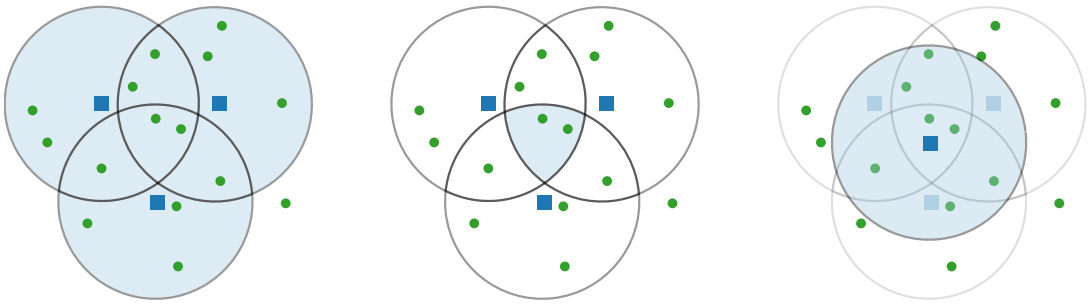


Figura 3.3: Una visualización de nuestro método. Primero obtenemos una representación de un solo valor brindada por la proyección multidimensional para cada registro en nuestro conjunto de datos, de tal forma nos permita ordenar los resultados en un arreglo por su similitud.



(a) Union. (b) Intersección. (c) Valor medio.

Figura 3.4: Ejemplo de nuestros enfoques para obtener los puntos mas cercanos utilizando tres puntos de consulta los cuales muestran diferentes resultados dependiendo del método empleado.

de consulta. Para llevar a cabo esta tarea, iteramos todos los pivotes que contienen los datos más similares y luego los trazamos siguiendo sus atributos espaciales (Figura 3.4a).

En nuestra segunda estrategia como puede ser vista en la Figua 3.4b, es recuperar los puntos comunes que están dentro de la consulta de rango de todos los puntos de consulta, ya que la estructura HashedCubes ya ordena los valores indexados, este enfoque no afecta drásticamente el rendimiento de nuestra estructura de datos.

Nuestro último enfoque genera un punto artificial al calcular el valor medio para todos los atributos de todo el conjunto de puntos de consulta como pude ser expresado en la Figura 3.4c. Una vez que se procesa este punto, realizamos una consulta de rango utilizando nuestro parámetro de umbral y recuperando todos los puntos contenidos.

3.5. Visualizando los datos

Para poder visualizar y comparar en un mapa geogrçáfico los puntos mas cercanos y más alejados segçún su similitud, utilizamos una paleta de colores para cada heatmap mostrado, desde el amarillo (más similar) al azul (menos similar). Por cada color de nuestra paleta, definimos un rango de consulta que va en aumento con respecto a la similitud, es decir el rango para los valores mas cercanos es menor que el rango de los más alejados similarmente. Esto es debido a que el heatmap con mayor rango contiene a los que están más cerca, es decir con un menor rango de búsqueda.

Esta visualización se puede observar en la Figura 3.5, como se puede observar, luego de combinar las diferentes capas con los heatmaps correspondientes a la paleta de colores, los puntos con tendencia cálida (amarilla, roja) representan a los datos más similares; por otro lado, los puntos con un color frio (morado, azul) representan a los puntos más lejanos similarmente.

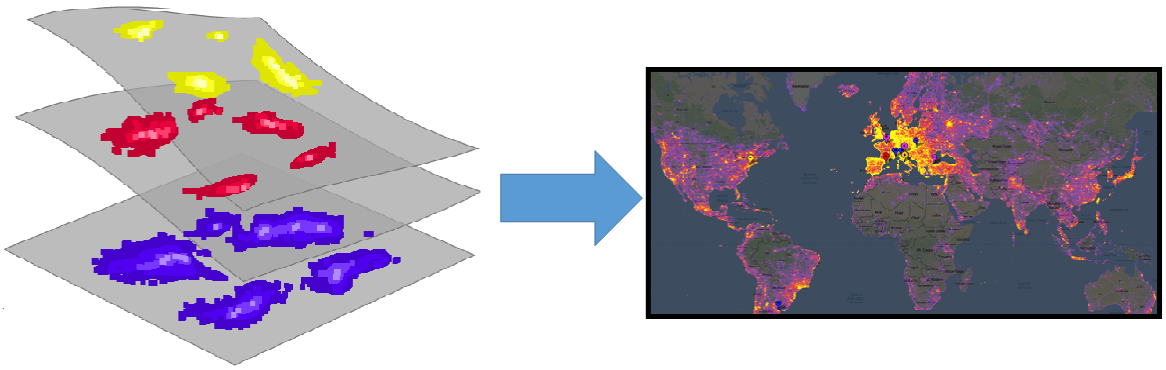


Figura 3.5: Utilizando tres paletas de colores para la visualización de datos por su similitud.

3.6. Implementación

Una arquitectura cliente-servidor es utiliza en nuestra implementación. Después de pre-calcular el mapeamiento multidimensional, este nuevo atributo particular de una dimensión se almacena junto con los otros datos para ser leídos por un bucle de eventos. El servidor es una implementación de C++ para explotar la operación de una porción de memoria asignada previamente para evitar y reutilizar las estructuras de datos existentes proporcionadas por STL entre otras bibliotecas. Explicar cada parte del marco y para qué sirve

Hemos creado una visualización de front-end para consultar los datos que se insertan utilizando Javascript, HTML5, SVG y D3.js. Considere la Figura 3.3 que visualiza la distribución de similitud de los conjuntos de datos de Brightkite, que consisten en 4.5 millones de registros que van desde abril de 2008 hasta octubre de 2010, en un mapa geográfico. Utilizamos un total de cinco mapas de calor con diferentes colores de paleta

en los que cada mapa de calor cubre progresivamente el área que contiene la mayoría de los elementos similares. Los porcentajes se basan en los valores que llena el usuario.

Para no interrumpir la interacción fluida, agregamos tres widgets en los que el usuario puede actualizar los parámetros de similitud que nuestro método desea mostrar, los describimos de la siguiente manera:

Selección de puntos

Agrega un número ilimitado de puntos con muchos cuadros de selección dependiendo del número de atributos. Estos puntos se envían al servidor para realizar la operación de consulta. El usuario también puede cambiar la operación de consulta múltiple dinámicamente.

Entrada para la consulta de rango

Toma cinco números enteros ordenados de menor a mayor. Estos cinco elementos representan los valores de consulta de rango que se muestran en el mapa con un mapa de calor del color, al lado del valor de consulta de rango ingresado. También incluimos una casilla de verificación para habilitar la densidad de datos para cada mapa de calor trazado.

Histograma de similitud

Se encuentra en la parte inferior de la interfaz. Este widget muestra de manera perceptiva la distribución de similitud de nuestro conjunto de datos explorado. Además, mostramos una barra de resaltado con el propósito de indicar dónde se ubican las consultas de puntos junto con un área azul. Representa la cantidad de datos cubiertos que se mostrarán en el mapa geográfico.

3.7. Consideraciones finales

En este capítulo se presentó como realizar la proyección a un espacio $1D$ con el objetivo de utilizarlo como pre-procesamiento para posteriormente aplicar consultas de similitud, evaluando los métodos considerados en el presente trabajo. En este sentido, teniendo una representación de los datos en una estructura, la recuperación de elementos contiguos similarmente es realizada mediante un mapa de calor como fue descrito anteriormente.

También fueron presentadas diferentes técnicas de recuperación de puntos, en el caso que la consulta involucra más de un punto.

Capítulo 4

Resultados

4.1. Consideraciones iniciales

En este capítulo se describen los experimentos computacionales realizados con la finalidad de analizar la visualización de similitud brindada por nuestro método. Los experimentos fueron divididos en dos etapas, cada cual con objetivos diferentes.

La primera detalla la diferencia en visualizar diferentes puntos de consulta sobre un mapa geográfico, el cual resulta en patrones de color diferentes para cada caso, según sea ajustado el rango de consulta. Nuestro segundo experimento evidencia la similitud utilizando nuestras tres técnicas propuestas para la recuperación de datos cuando la consulta involucra más de un punto. Con este objetivo evaluaremos nuestro método en relación a tres conjuntos de datos disponibles públicamente.

Además, mostramos la diferencia entre usar los tres métodos para combinar la operación de consulta múltiple. En la Tabla 4.1, resumimos la información relevante para proyectar e indexar el conjunto de datos en los conjuntos de datos descritos anteriormente. El número total de registros E, el valor máximo que tendría los datos proyectados utilizando las proyecciones *Z-order curve* y *Hilbert curve*, se expresan en las siguientes dos columnas. La siguiente columna representa el número de "pivotes" creados al indexar los valores proyectados y, finalmente, una breve descripción del esquema de cada conjunto de datos.

4.1.1. Cambiando parámetros de búsqueda

Nuestro primer resultado es el uso de un conjunto de cuatro millones de usuarios en el conjunto de datos Brightkite (Cho et al., 2011). Los datos en bruto constan de cinco dimensiones: Usuario, Fecha, Hora, Latitud y Longitud. En la Figura 4.1 podemos visualizar la distribución de similitud de la indexación de la Fecha y la Hora en la ubicación (es decir, Lat y Lon). Usando la consulta de puntos de hora del día

Conjunto de datos	Objetos(E)	Z-order	Hilbert	Pivotes	Esquema
brightkite	4,5M	317	299	168	lat, lon, hour of the day(24), day of month(31)
NYPD incident	415901	427	508	197	lat, lon, jurisdiction for incident(18), classification code(16)
Twitter	210,6M	751	695	131	lat, lon, app(4), device(5), language(15)

Cuadro 4.1: Descripción de los conjuntos de datos considerados para obtener nuestros resultados, junto con la descripción de recursos al indezar sus valores categóricos para consultas de similitud.

(valor de 9) y día de la semana (valor del martes), los elementos más similares se encuentran en los lados de Estados Unidos (lados este y oeste), la mayoría de los datos se encuentran en Europa, específicamente en el Reino Unido, Países Bajos y Alemania (de una paleta de colores de rojo a azul como se muestra en la esquina superior derecha). Además, podemos observar una distribución uniforme de los datos en el histograma de similitud en la parte inferior de la interfaz donde se resaltan dos grupos bien separados por lo es posible visualizar este fenómeno en el mapa. Como resultado, la mayoría del rango de colores del mapa de calor está presente en el mapa.

Nuestro próximo conjunto de datos también tiene cuatro millones de entradas de quejas civiles de la ciudad de Nueva York. Los datos de entrada, que se pusieron a disposición como archivos csv, tienen entradas de 415901. En este ejemplo, mostramos cuatro puntos de consulta diferentes (Figura 4.2) en los que podemos ver la distribución de similitud en el grupo dependiendo de dónde se encuentra la consulta de puntos, como podemos ver en el histograma sobre cada imagen, más la similitud Los grupos están más separados, lo que resulta en interesantes patrones de color. Por ejemplo, dada la paleta de colores en la esquina superior derecha de cada imagen, el punto de consulta está justo en el grupo de similitud con la mayor cantidad de datos. Podemos observar este fenómeno en el histograma de similitud en la parte inferior de la interfaz, por lo tanto, el color del mapa de calor trazado es principalmente el color de los rangos más cercanos, lo que significa que podemos visualizar los elementos más similares con un color cálido(Figura 4.2a). En el siguiente caso de la Figura 4.2b, el punto de consulta se encuentra en medio de dos grupos de similitud, es por esta razón que el verde de la paleta de colores ha aparecido alrededor del mapa por lo que podemos confirmar que nuestra consulta involucra los datos a una distancia media similarmente. El mismo criterio ocurre en la Figura 4.2c debido a que el punto ahora está más alejado de cualquier grupo, el mapa de color con la predominancia verde aparece en más regiones dando a conocer las regiones donde los datos pertenecientes a los grupos disímiles están

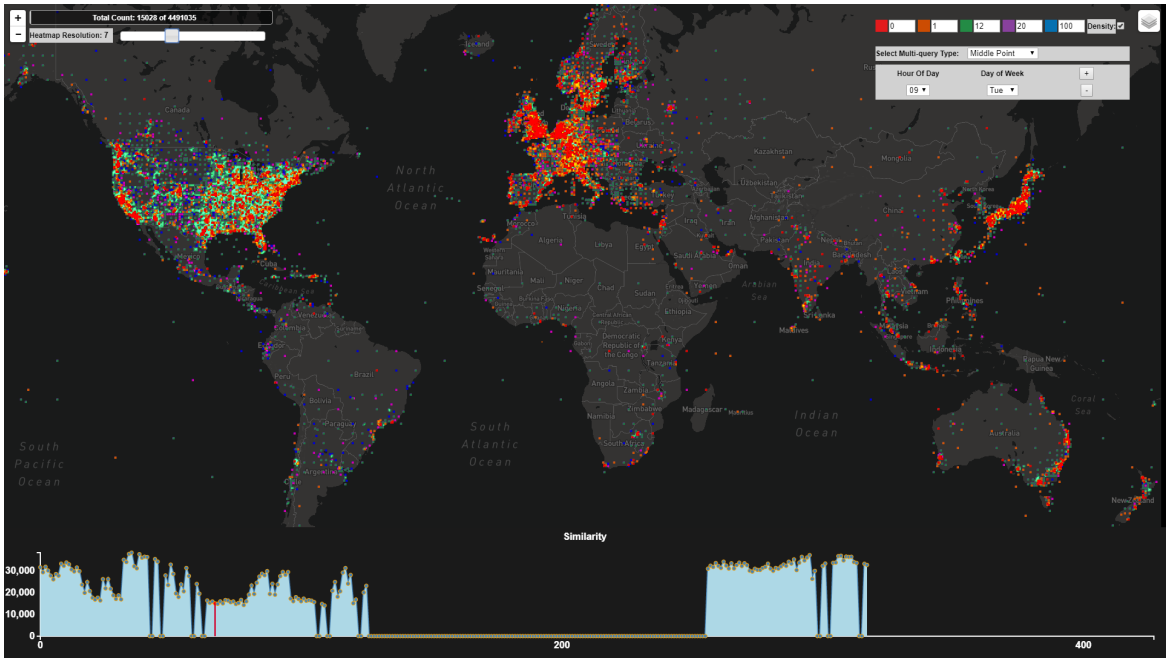


Figura 4.1: Exploring the Brightkite checking similarity distribution using five different colormaps that enables a visual differentiation of the similarity over a geographical map.

ubicados. De otra forma, en nuestro último ejemplo (Figura 4.2d), el mapa de calor de los elementos menos similares es más relevante ya que la consulta de puntos está más aislada del resto del conjunto de datos.

4.1.2. Cambiando método de recuperación de datos

Nuestro último conjunto de datos tiene tweets recopilados de los Estados Unidos durante las fechas de noviembre de 2011 a junio de 2012. La cantidad total de tweets geolocalizados es de aproximadamente 210 millones y cada tweet consta de una información espacial; dispositivo, aplicación e idioma utilizados como información categórica, respectivamente 5, 4 y 15 valores distintos.

En este ejemplo, usamos el índice para mostrar la diferencia en la aplicación de nuestros tres métodos implementados, es decir, Unión, Intersección y Valor medio (Figura 3.4. A continuación mostraremos el resultado de tres mapas de colores diferentes utilizando los tres diferentes métodos de recuperación para consulta con múltiples puntos con los mismos puntos de consulta.

Usaremos tres puntos para la consulta. El primer punto de consulta se refiere a un valor “Ninguno” para Idioma, Dispositivo y Aplicación. El siguiente punto de consulta tiene un valor de idioma de Ïtaliano”, “Android” para el dispositivo, y el valor “Instagram”para la aplicación. El punto de consulta final tiene un valor “ruso” para el idioma, el dispositivo es un “iPad” y se envía a través de la aplicación “twitter”. El ejemplo de la “Unión”, en la Figura 4.3 la mayor parte del mapa posee un color

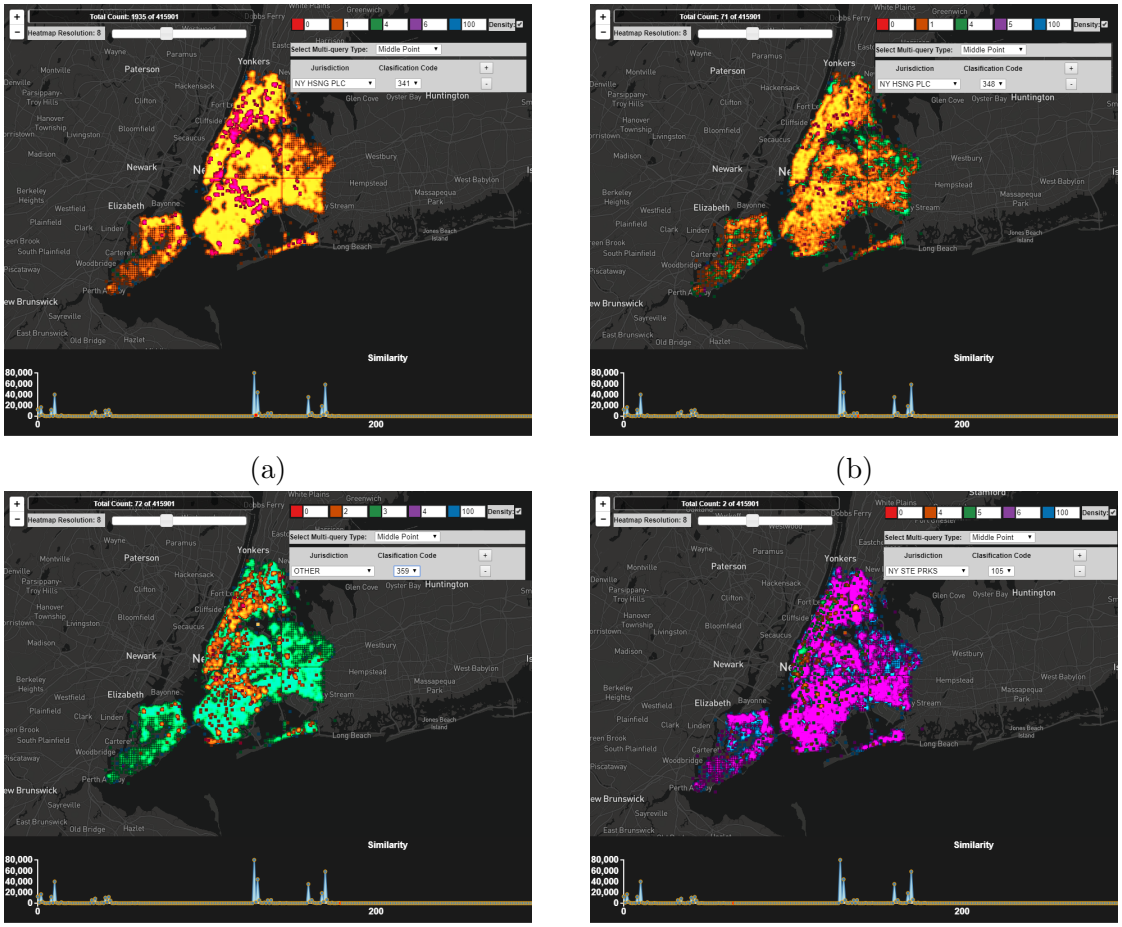


Figura 4.2: Exploración visual del conjunto de datos de quejas civiles de la Ciudad de Nueva York utilizando cuatro puntos de consulta y consultas de rango diferentes.

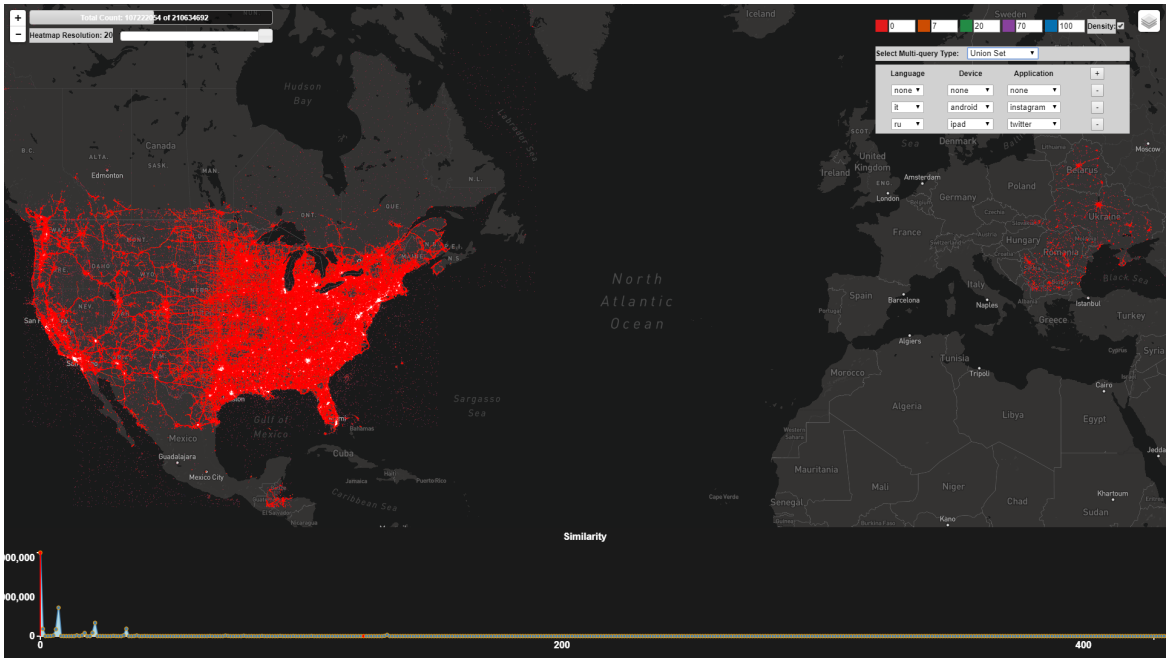


Figura 4.3: Visualizando 210 millones de usuarios de Twitter geolocalizados públicamente. Método Union

rojo, esto es debido a que el resultado de la operación "Unión" de los puntos descritos anteriormente contiene en su mayoría a todo el conjunto de datos. Sin embargo, en el caso de la "Intersección", apreciamos un dominio púrpura en el mapa de la Figura 4.4, por lo que se puede concluir que los datos están lejos de la intersección específica entre puntos y rango. Finalmente, usando un "punto medio" para la recuperación de datos tenemos dos secciones separadas visualmente, entre distribuidas utilizando los colores naranja y rojo, lo que muestra que todos los puntos están más cerca del punto de consulta obtenido por el hecho de usar colores cálidos, en la distribución del mapa se puede observar que en la parte este de Estados Unidos se encuentran los valores más cercanos al punto medio resultante.

4.2. Consideraciones finales

En este capítulo se presentó diferentes experimentos comparativos para las visualizaciónes por similitud brindadas por nuestro método. Nos concentramos, principalmente, en dos enfoques: Cambiando los parámetros de búsqueda (punto y rango), y cambiando el método de recuperación de datos (unión, intersección, y punto medio).

Finalmente, los resultados muestran que la represencación de mapas de calor por los diferentes tipos de consulta evaluados tienen gran importancia al interpretar los datos y es de vital importancia en la visualización resultante.

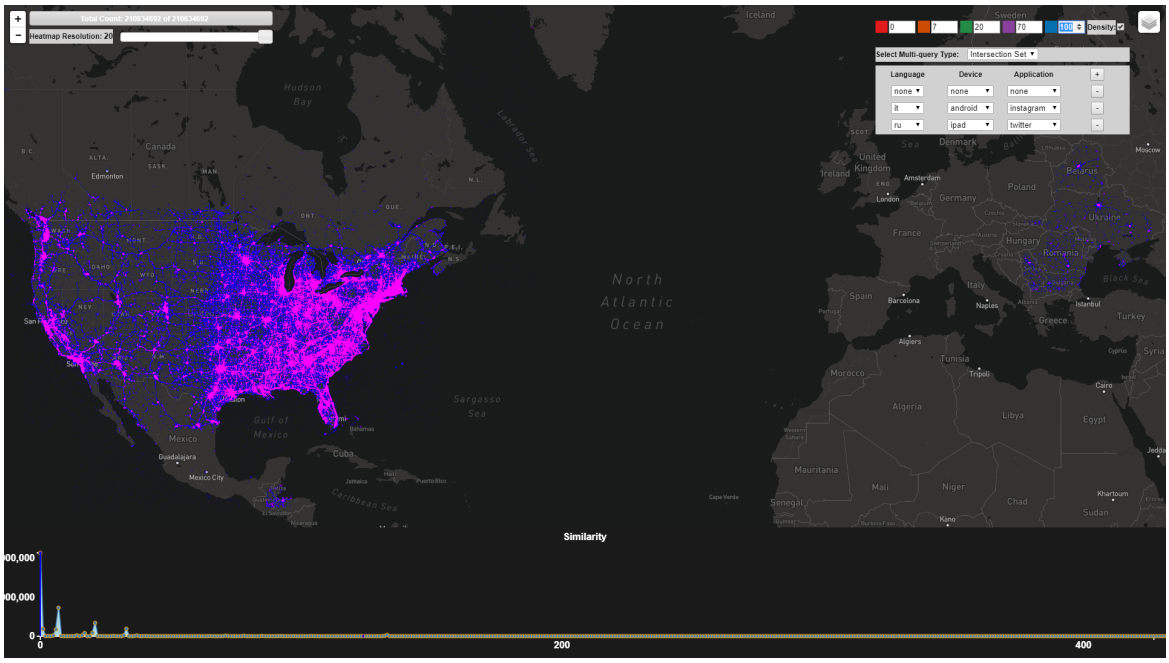


Figura 4.4: Visualizando 210 millones de usuarios de Twitter geolocalizados públicamente. Método intersección.

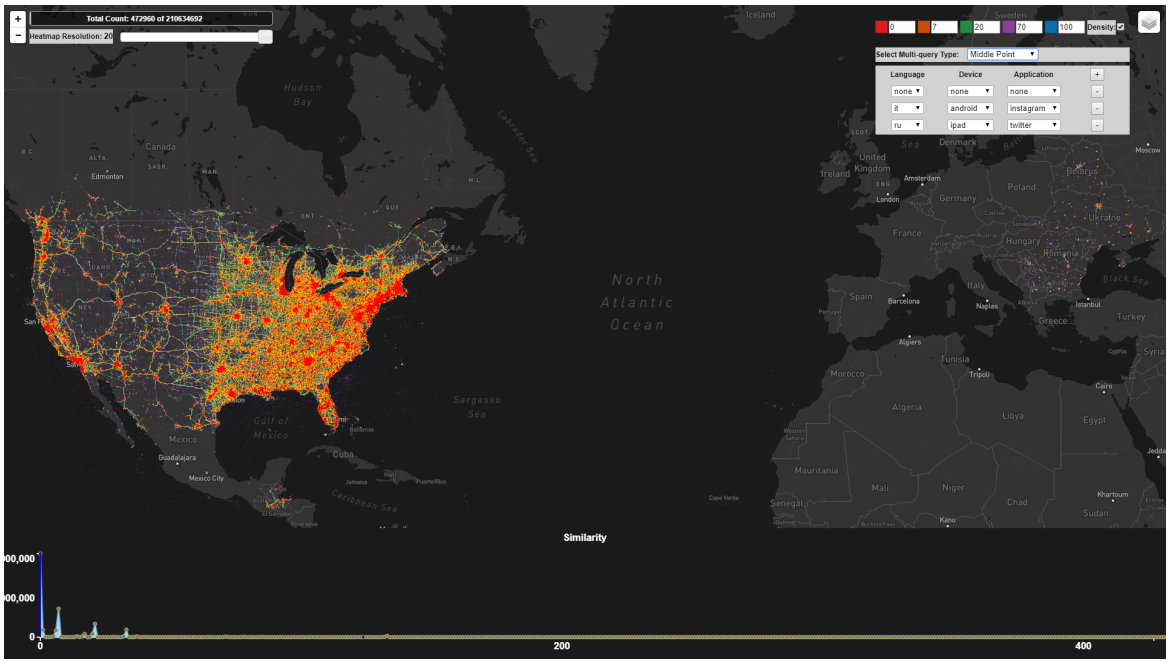


Figura 4.5: Visualizando 210 millones de usuarios de Twitter geolocalizados públicamente. Método valor medio.

Capítulo 5

Conclusiones

5.1. Discusión y Limitaciones

A nuestro conocimiento, ninguno de los métodos del estado del arte en la visualización de modelos exploratorios ofrece una visualización de similitud sobre grandes conjuntos de datos espacio temporales, multidimensionales. Nuestro método propuesto es capaz de trazar una distribución de similitud de un conjunto de datos que proporciona un conjunto de puntos de consulta en un mapa geográfico, como se ve en Capítulo 4. También hemos demostrado su valor mostrando tres casos diferentes con una evaluación de nuestras técnicas de recuperación con conjuntos de datos públicos. Sin embargo, todavía hay muchas limitaciones y oportunidades de mejora.

Proyectar datos en un espacio unidimensional reduce la complejidad de tratar con un conjunto de datos de alta dimensión. Además, solo utilizamos las proyecciones de *Z-order* y *Hilbert curve* para proyectar atributos categóricos descartando otros tipos de dimensión, como los atributos temporales o espaciales. A pesar de que podemos usar atributos categóricos, sólo es posible para aquellos con naturaleza ordinal debido a que necesitamos establecer la medida de distancia entre cualquier par de elementos.

Además, al obtener el valor proyectado resultante para cada elemento en el conjunto de datos, se obtiene un valor entero que va de cero a un número que es proporcional al número de atributos categóricos y los valores posibles de cada uno de estos atributos. Por ejemplo, si el conjunto de datos explorado contiene muchos atributos categóricos y cada uno de estos atributos puede tomar un alto número de valores posibles, la proyección resultante expandirá todo el conjunto de datos proyectado. Además, aumentará el número de pivotes generados por un HashedCubes, lo que tendrá un impacto negativo tanto en el uso de la memoria como en el tiempo de ejecución.

Finalmente, la elección entre el *Z-order* y *Hilbert curve* que utiliza nuestro método depende de la distribución general de los valores de nuestro conjunto de datos. Si todas las instancias del conjunto de datos están bien descritas dentro de un cuadrado descrito precisamente por una descripción Z, entonces *Z-order curve* es nuestra mejor opción;

de lo contrario, a medida que se produzcan saltos más largos en instancias continuas del conjunto de datos, mayor será la precisión de la proyección de la curva de *Hilbert*. Esta selección se basa en la preferencia del analista para decidir qué método de proyección usar.

5.2. Conclusión

En esta tesis, presentamos una estructura de datos novedosa que combina HashedCubes y métodos de reducción de dimensionalidad en $1D$ para proporcionar un conjunto de consultas de similitud para realizar en conjuntos de datos muy grandes. Como resultado de la combinación mencionada anteriormente, nuestro método también comparte sus características. Como la escalabilidad con un gran número de dimensiones, el rendimiento en términos de tiempo de consulta y la imposibilidad de actualizar la estructura de datos una vez que se construye. Sin embargo, incluimos una característica esencial para aumentar el rango de consultas a realizar.

Actualmente, para determinar la función de transferencia de color de cada mapa de similitud, proporcionamos un panel que muestra la paleta de colores que se usa para cada consulta y una escala de colores que está oculta para el usuario. Con la finalidad de proporcionar una interacción más perceptiva, creemos que un widget para establecer estos valores de forma dinámica tendría un impacto positivo en nuestro marco implementado.

Bibliografía

Nyc tlc trip data. http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml.

Amato, F., De Santo, A., et al. (2015). Semtree: An index for supporting semantic retrieval of documents. In *Data Engineering Workshops (ICDEW), 2015 31st IEEE International Conference on*, pages 62–67. IEEE.

Bellman, R., Bellman, R., et al. (1957). *Dynamic Programming*. Rand Corporation research study. Princeton University Press.

Beyer, K. S., Goldstein, J., et al. (1999). When is "nearest neighbor" meaningful? In *Proceedings of the 7th International Conference on Database Theory*, ICDT '99, pages 217–235, London, UK, UK. Springer-Verlag.

Böhm, C., Berchtold, S., et al. (2001). Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Comput. Surv.*, 33(3):322–373.

Cho, E., Myers, S. A., et al. (2011). Friendship and mobility: User movement in location-based social networks. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '11, pages 1082–1090, New York, NY, USA. ACM.

Chung, F. R. K. (1996). *Spectral Graph Theory (CBMS Regional Conference Series in Mathematics, No. 92)*. American Mathematical Society.

Doraiswamy, H., Vo, H. T., et al. (2016). A gpu-based index to support interactive spatio-temporal queries over historical data. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*, pages 1086–1097.

Faloutsos, C. y Lin, K.-I. (1995). Fastmap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. *SIGMOD Rec.*, 24(2):163–174.

Gilbert, E. N. (1958). Gray codes and paths on the n-cube. *The Bell System Technical Journal*, 37(3):815–826.

Gray, J., Chaudhuri, S., et al. (1997). Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data Mining and Knowledge Discovery*, 1(1):29–53.

Hubert, D. (1891). Ueber die stetige abbildung einer linie auf ein flächenstück. *Mathematische Annalen*, 38:459–460.

Jolliffe, I. (2011). *Principal component analysis*. Springer.

Lins, L., Klosowski, J. T., et al. (2013). Nanocubes for real-time exploration of spatiotemporal datasets. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2456–2465.

Maaten, L. v. d. y Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605.

Martins, R., Minghim, R., et al. (2015). Explaining neighborhood preservation for multidimensional projections. In *CGVC*, pages 7–14.

Miranda, F., Lins, L., et al. (2017). Topkube: a rank-aware data cube for real-time exploration of spatiotemporal data. *IEEE Transactions on Visualization and computer graphics*.

Morton, G. M. (1966). A computer oriented geodetic data base and a new technique in file sequencing.

Pahins, C. A., Stephens, S. A., et al. (2017). Hashedcubes: Simple, low memory, real-time visual exploration of big data. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):671–680.

Wang, Z., Ferreira, N., et al. (2017). Gaussian cubes: Real-time modeling for visual exploration of large multidimensional datasets. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):681–690.

Zhou, W., Yuan, C., et al. (2013). Large scale nearest neighbors search based on neighborhood graph. In *2013 International Conference on Advanced Cloud and Big Data*, pages 181–186.